UNIVERSITE DE NANCY I

CENTRE DE RECHERCHE EN INFORMATIQUE DE NANCY

METHODES DE FILTRAGE EQUATIONNEL ET DE PREUVE AUTOMATIQUE DE THEOREMES

THESE DE DOCTORAT DE L'UNIVERSITE DE NANCY I
Specialité INFORMATIQUE
PRESENTEE PAR

Jalel MZALI

SOUTENUE LE 27 octobre 1986

DEVANT LA COMMISSION D'EXAMEN

PRESIDENT	R. MOHR
RAPPORTEURS	J. HSIANG
	R. MOHR
EXAMINATEURS	J.P. JOUANNAUD
	C. KIRCHNER
	P. LESCANNE
	J.L. REMY

UNIVERSITE DE NANCY I

CENTRE DE RECHERCHE EN INFORMATIQUE DE NANCY

METHODES DE FILTRAGE EQUATIONNEL ET DE PREUVE AUTOMATIQUE DE THEOREMES

THESE DE DOCTORAT DE L'UNIVERSITE DE NANCY I
Specialité INFORMATIQUE
PRESENTEE PAR

Jalel MZALI

SOUTENUE LE 27 octobre 1986

DEVANT LA COMMISSION D'EXAMEN

PRESIDENT	R. MOHR
RAPPORTEURS	J. HSIANG
	R. MOHR
EXAMINATEURS	J.P. JOUANNAUD
	C. KIRCHNER
	P. LESCANNE
	J.L. REMY

METHODES DE FILTRAGE EQUATIONNEL ET DE PREUVE AUTOMATIQUE DE THEOREMES

Thèse de doctorat de l'université de Nancy 1

Spécialité Informatique

présentée le 27 Octobre 1986 par

Jalel MZALI

Devant la commission d'examen

R. Mohr, Professeur à L'Institut National Polytechnique de Lorraine (Président)

J. Hsiang, Professeur à la State University of New York at Stony Brook

J.P. Jouannaud, Professeur à l'université Paris Sud, Orsay

C. Kirchner, Chargé de recherche au CNRS

P. Lescanne, Chargé de recherche au CNRS

J.L. Rémy, Chargé de recherche au CNRS

RESUME

L'objet de la première partie de cette thèse est l'étude et l'implantation de différentes méthodes de démonstration automatique basées essentiellement sur un algorithme de complétion rapide appelé SKB et un algorithme de complétion sans échec appelé UKB. SKB est un algorithme de complétion qui privilégie la règle de simplification par rapport à celle de superposition, nous étudions cet algorithme et son implantation, nous prouvons sa correction totale par la technique des *ordres sur les preuves équationnelles* de Bachmair-Dershowitz-Hsiang. L'algorithme UKB de complétion sans échec, dû à Hsiang-Rusinowitch est étudié et implanté, il nous permet de construire des preuves dans les théories équationnelles et les théories équationnelles généralisées.

La deuxième partie de cette thèse est consacrée à l'étude du problème de filtrage qui est crucial pour la simplification et la réécriture des termes. Ce problème est étudié d'un point de vue algébrique, ce qui nous permet d'exprimer les principales phases des algorithmes de filtrage par des règles d'inférence simples dont la traduction sous forme d'un système de réécriture fournit un algorithme de filtrage. Cette étude algébrique nous permet de donner un algorithme original dans le cas du filtrage modulo la distributivité et par conséquent de prouver la décidabilité du filtrage distributif. Un algorithme pour le filtrage modulo l'axiome de commutativité droite est également donné. De la même manière que les transformations d'équations, nous définissons les transformations de théories dans lesquelles les équations sont résolues. Dans le cas d'union de théories E∪T, nous donnons des conditions suffisantes pour ramener le problème de E∪T-filtrage d'un système d'équations au problème de E-filtrage (ou T-filtrage) d'un autre système d'équations.

Mots clefs: Théorie équationnelle, preuve automatique, preuve par réfutation, algorithme de complétion, algorithme de complétion sans échec, filtrage équationnel, décomposition, fusion, mutation.

À la mémoire de mon père,
à ma mère
et à Sarra.

REMERCIEMENTS

Je remercie sincèrement tous les membres du jury.

Jieh Hsiang, Professeur à la State University of New York, m'a initié aux preuves de théorème, il m'a acceuilli dans son équipe à Stony Brook où nous avons développé le système SBREVE. Sans son amicale pression, sa compétence et sa disponibilité, cette réalisation n'aurait pas vu le jour. Je lui dois également d'avoir pu travailler dans d'excellentes conditions matérielles. Ses conseils et ses encouragement m'ont été très précieux et c'est un plaisir de pouvoir lui exprimer ma profonde reconnaissance.

Ce travail a été réalisé sous la direction de Jean-Pierre Jouannaud, Professeur à l'université Paris Sud Orsay, dont les conseils et les remarques constructives ne m'ont jamais fait défaut. Je lui dois beaucoup pour m'avoir depuis longtemps soutenu et encouragé. Il m'a initié à la recherche depuis le DEA, aidé à développé mes connaissances et à formaliser mes idées parfois embryonnaires, avec une grande rigueur et ouverture d'esprit. Je lui adresse ici mes sincères remerciements.

Claude Kirchner, chargé de recherche au CNRS, a lui aussi dirigé une grande partie de cette thèse, sa grande expérience de concepteur au cours de la réalisation de REVE3, ses critiques très constructives et amicales qui ne font jamais défaut et son inlassable dynamisme m'ont été très précieux.

Pierre Lescanne, chargé de recherche au CNRS, m'a donné l'exemple d'un chercheur qui est aussi programmeur. Il m'a aidé non seulement en ce que toutes les concrétisations de mes recherches soit dans le cadre du projet REVE qu'il a lancé, mais aussi par ses encouragements à développer à la fois résultats théoriques et implantations.

Roger Mohr, Professeur à l'Institut Polytechnique de Loraine, a bien voulu apporter ses compétences en intelligence artificielle et me fait l'honneur de présider ce jury. Je lui suis reconnaissant de la confiance qu'il m'a signifiée ces dernières années.

Jean-Luc Rémy, chargé de recherche au CNRS, a toujours suivi avec intérêt mes recherches, il n'a jamais manqué de me faire profiter de ses idées, de ses remarques constructives et de son amical soutien.

Je remercie tout autant mes professeurs de l'université de Tunis qui m'ont donné les premières bases mathématiques necessaires pour ce travail.

Cette thèse a été réalisée au sein de l'équipe EURECA du CRIN et je tiens à remercier tous ceux qui de près ou de loin ont facilité cette réalisation en créant une ambiance de travail efficace et agréable. Je tiens à remercier particulièrement Michael Rusinowitch pour les fructueuses conversations que nous avons eu ensemble au cours de la réalisation de SBREVE, et pour les critiques amicales concernant la rédaction de cette thèse, Hélène Kirchner pour avoir relu la première version de ce travail et prodigué remarques et critiques constructives, Pierre Marchand pour m'avoir fait profiter de sa profonde culture scientifique, Jean-Claude Derniame et Jean-Pierre Finance pour m'avoir introduit à l'informatique, Djemel Ziou, Pierre Rety et tous les membres de l'équipe pour leur amical apui.

Je ne manquerai pas non plus de remercier toute l'équipe de Stony Brook, particulièrement Gene Stark. Je remercie également le GRECO de programmation du CNRS pour le soutien matériel qui m'a permis de mener à terme ce travail.

J'adresse mes remerciements chaleureux à tous mes proches et à toutes les personnes qui m'ont aidé et soutenu tout au long de ce travail, Je remercie particulièrement Faiza Slama pour ses encouragements qui ne m'ont jamais fait defaut, Fairouz Slama pour son aide à la rédaction de l'article "Matching With Distributivity" et tous mes amis pour leurs Unfailing... Good Humor!

Mes derniers remerciements mais non les moindres vont à Sarra, autant pour les nombreuses discussions techniques et non-techniques, que pour son précieux soutien moral et ses encouragements.

References

INTRODUCTION

L'objet de la première partie de cette thèse est l'étude et l'implantation de différentes méthodes de démonstration automatique basées essentiellement sur des extensions d'un algorithme original de complétion plus rapide que l'algorithme classique de Huet.

La seconde partie de ce travail est consacrée à une étude algèbrique du problème de filtrage équationnel, qui permet de décrire des algorithmes de filtrage dans un formalisme trés général.

La procédure de complétion de système de réécriture a été introduite par Knuth-Bendix pour résoudre le problème du mot en algèbre universelle [Knuth-Bendix-70]. Elle permet de transformer un ensemble d'équations en un ensemble d'équations orientées appelées règles de réécriture de telle sorte que le résultat du calcul ne dépende pas du choix des règles utilisées (ce qu'on appelle propriété de confluence). Cette procédure peut être vue comme un moyen de compiler une spécification équationnelle E, pour laquelle le calcul est non déterministe, en un ensemble de règles de réécriture R

qui engendre la même équivalence sur les termes que l'ensemble d'équations E de départ. Pour le problème d'égalité des termes, cet ensemble de règles fournit une procédure de décision à condition que la relation de réécriture soit noethérienne (sans chaîne de réécriture infinie). En effet pour prouver que deux termes sont E-égaux il suffit alors de calculer leurs formes irréductibles dans R et de vérifier leur identité (égalité syntaxique). Notons que cette méthode est beaucoup plus avantageuse que la méthode de remplacement des égaux par des égaux utilisant les équations de E et nécessitant un algorithme de retour arrière (backtracking) qui est fort coûteux et ne garantit pas un succès dans tous les cas.

Pour construire un système R de règles qui soit convergent (c'est-à-dire confluent et noethérien) à partir d'un ensemble d'équations E, la procédure de complétion de Knuth-Bendix consiste à ajouter au système R, toutes les paires de termes de taille minimale qui proviennent de la réécriture d'un terme de deux façons différentes, (on appelle ces paires des paires critiques), et à les orienter (en utilisant un ordre bien fondé, de manière à conserver la propriété de terminaison). Ce processus doit être itéré jusqu'à ce que, le cas échéant, la procédure s'arrête en fournissant un système convergent.

Durant ces dix dernière années, la procédure de Knuth-Bendix s'est avérée un outil puissant pour une large classe de problèmes: preuves inductives [Musser-80,Goguen-80,Huet-Hullot-80,Jouannaud-Kounalis-86], démonstration automatique de théorèmes [Peterson-83,Hsiang-82,Fribourg-83], synthèse de programmes [Hsiang-Plaisted-82,Dershowitz-84], exécution de spécifications équationnelles [Goguen,etal.-82]...

Récemment, plusieurs auteurs ont donné des méthodes pour améliorer l'efficacité de la procédure de complétion [Winkler-Buchberger-83,Kuechlin-85,Kapur,etal.-85]. Ces méthodes sont toutes basées sur le principe de la limitation du calcul des paires critiques.

Egalement dans le but d'en ameliorer l'efficacité, nous exposons dans cette thèse une procédure de complétion où la notion de simplification est privilégiée par rapport à la génération des paires critiques. Nous pensons en effet que c'est la simplification qui apporte toute sa puissance aux systèmes de réécriture:

- Lorsqu'on génère une paire critique, on augmente le système d'une équation; lorsqu'on simplifie, on remplace le système par un système plus simple.
- La génération de paires critiques utilise un algorithme d'unification, la simplification utilise un algorithme plus rapide de filtrage.

Afin de justifier le mécanisme implanté, nous utilisons le formalisme des ordre sur les preuves équationnelles proposé par Bachmair-Dershowitz-Hsiang[Bachmair,etal.-86] Après la preuve de correction de notre algorithme dans ce formalisme, nous en décrivons ensuite l'implantation SBREVE dans le cadre du système REVE.

Dans le cas où toutes les paires critiques peuvent être orientées, la procédure de complétion peut ne pas terminer (on dit qu'elle diverge). Dans ce cas on ne sait à aucun momment si elle n'a pas encore trouvé de système convergent ou bien s'il n'existe pas de tel système qui soit fini. Cette divergence de l'algorithme n'est toutefois pas si catastrophique que

l'on pourrait le croire: Huet [Huet-81] a montré que la procédure de Knuth-Bendix, dans ce cas de divergence, reste une procédure de semi-décision pour l'égalité de deux termes t et t'. Si $t=_E t'$ alors il existe une itération de l'algorithme à partir de laquelle t et t' auront même forme normale. il suffit donc de calculer à chaque étape les formes normales de t et t' et de vérifier leur égalité.

Nous présentons une extension de cette méthode due à Hsiang et Rusinowitch [Hsiang-Rusinowitch-85b], appelée UKB, (pour Unfailing Knuth-Bendix) qui enlève la restriction de noetherianité de la réécriture, mais conserve la propriété d'être une procédure de semi-décision pour le problème du mot dans les théories équationnelles. On peut donc l'utiliser comme procédure de semi-décision même dans le cas d'existence d'équations non-orientables. Nous décrivons son implantation et ses applications aux preuves par réfutation.

Le filtrage est l'élément central de la simplification, une règle $g \rightarrow d$ simplifie un terme t[s], lorsqu'il existe un filtre σ tel que $\sigma g = s$. Cette égalité devient égalité modulo un ensemble d'axiomes E ($\sigma g =_E s$) lorsque nous consirérons la réécriture modulo. La recherche des substitution σ telles que $\sigma M =_E N$ est appelé problème de filtrage équationnel. Le filtrage équationnel est à la base de la réécriture de termes dans l'agorithme de complétion équationnel (ou modulo) [Jouannaud-Kirchner-84], il est aussi à la base de la réduction des termes dans les langages basés sur la réécriture comme OBJ [Futatsugi,etal.-85].

Burckert [Burckert-86] a récemment montré que le filtrage n'est pas, comme on pourrait le croire, un cas particulier de l'unification ($\sigma M =_E \sigma N$). Ramener un problème de filtrage à un problème d'unification en renomant les variables du deuxième terme N de l'équation à résoudre nécessite l'introduction de nouvelles constantes, et cela peut rendre la théorie (E union l'ensemble des nouvelles constantes) indécidable!

Dans la deuxième partie de cette thèse, nous étudions le problème du filtrage équationnel en définissant des opérations élémentaires sur les équations et les systèmes d'équations à résoudre. Nous montrons comment les principales opérations, définies dans le cas de l'unification par C. Kirchner [Kirchner-85], peuvent s'exprimer par des règles d'inférence simples dont la traduction en un système de réécriture fourni directement un algorithme de filtrage.

Cette étude algébrique nous permet de donner un algorithme dans le cas du filtrage modulo l'axiome de distributivité et nous prouvons donc la décidabilité du filtrage distributif, la décidabilité de l'unification distributive étant encore un problème ouvert. Un algorithme pour le filtrage modulo l'axiome de commutativité droite est aussi donné.

Nous décrivons également des transformations d'équations de filtrage qui simplifient non pas l'équation elle-même, mais la théorie dans laquelle l'équation doit être résolue. Dans le cas d'union de théories $E_1 \cup E_2$, nous donnons des conditions suffisantes pour ramener le problème de $E_1 \cup E_2$-filtrage d'un système d'équations au problème de E_1-filtrage (ou de E_2-filtrage) d'un autre système d'équations.

PARTIE I

1. **Algèbres des Termes**

1. Algèbres et théories équationnelles

1.1. Structure d'algèbre

Soit F un ensemble dénombrable de **symboles de fonction**, et soit **ar** une application de F dans $\mathbb{N}$ qui à chaque élément f de F associe un entier ar(f) appelé arité de f. F est partitionné en une réunion d'ensemble F_i

$$F_i = \{f \in F,\ ar(f) = i\}$$

Les symboles de fonction d'arité 0 sont appelés **constantes** et notés a, b, c... et ceux d'arité non nulle sont désignés par les lettres f, g, h...

Définition 1.1 -- Une **F-algèbre** est un couple $M=(D_M,F_M)$ où D_M est un ensemble non vide appelé domaine de M, et F_M une famille d'opérations sur D_M indexées par l'ensemble des symboles de fonction de F. Il existe une application de F dans F_M qui à f associe f_M. Si ar(f)=k, f_M est une application de D_M^k dans D_M. On note $F_M = \{f_M \mid f \text{ appartient à } F\}$.

○

Exemple 1.1 -- Soit F = {|, &} un ensemble de symboles de fonction binaire (d'arité égale à deux). Nous choisissons comme domaine $\mathbb{B} = \{0,1\}$, et comme opérations $|_{\mathbb{B}}$ et $\&_{\mathbb{B}}$ définies comme suit :

$\|_{\mathbb{B}}$	0	1
0	0	1
1	1	1

$\&_{\mathbb{B}}$	0	1
0	0	0
1	0	1

Le couple $(\mathbb{B}, \{|_{\mathbb{B}}, \&_{\mathbb{B}}\})$ est une F-algèbre.

∇

1.2. Opérations sur les algèbres

Définition 1.2 -- Soit $A=(D_A,F_A)$ et $B=(D_B,F_B)$ deux F-algèbres. Un **homomorphisme** de F-algèbre (ou F-homomorphisme) ϕ de A dans B est une application de D_A dans D_B, telle que pour tout symbole de fonction f d'arité n dans F et pour tout élément $\delta_1,..,\delta_n$ dans D_M, on ait :

$$\phi(f_A(\delta_1,..,\delta_n)) = f_B(\phi(\delta_1),..,\phi(\delta_n))$$

Un homomorphisme d'une F-algèbre dans elle même est appelé **endomorphisme**. si de plus il est bijectif, il est appelé **automorphisme**. Un homomorphisme bijectif est appelé **isomorphisme**.

○

Exemple 1.2 -- Soit la F-algèbre sur $(\mathbb{N}, \{+_{\mathbb{N}}\})$ où $\mathbb{N}$ est l'ensemble des entiers naturels et $+_{\mathbb{N}}$ l'addition dans $\mathbb{N}$. F est donc l'ensemble qui contient le symbole de fonction + d'arite 2. Soit $(\mathbb{B},+_{\mathbb{B}})$ la F-algèbre de même domaine $\mathbb{B}$ que l'exemple précédent, et $+_{\mathbb{B}}$ définie par :

$+_{\mathbb{B}}$	0	1
0	0	1
1	1	0

Considérons ϕ l'application de $\mathbb{N}$ dans $\mathbb{B}$ qui à un entier n fait correspondre 0 si n est pair et 1 sinon. ϕ est un F-homomorphisme.

∇

Définition 1.3 -- Soit $M=(D_M,F_M)$ une F-algèbre. Une F-algèbre $N=(D_N,F_N)$ est dite une **sous-algèbre** de M si et seulement si D_N est inclus dans D_M et pour tout f de F, la restriction de f_M à D_N coincide avec f_N.

○

Définition 1.4 -- Soit une famille de F-algèbres $M_i=(D_{M_i},F_{M_i})$ indexées par un ensemble I. La F-algèbre **produit** M est la F-algèbre

$$(\prod_{i\in I} D_{M_i}, \pi)$$

où π est l'ensemble des opérations sur $\prod_{i\in I} D_{M_i}$ associées aux éléments de F. Pour un élément f de F et pour $(\delta_1,..,\delta_n)$ un élément du produit ensembliste des D_{M_i}, nous avons

$$f_M(\delta_1,..,\delta_n) = (f_{M_1}(\delta_1),..,f_{M_n}(\delta_n))$$

○

1.3. Algèbre libre

Définition 1.5 -- Soit K une classe quelconque de F-algèbres et X un ensemble. Nous appelons **F-algèbre K-libre engendrée par X** (ou sur X) toute F-algèbre $M=(D_M, F_M)$ telle que:

1) $M \in K$

2) $D_M \supseteq X$

3) pour toute F-algèbre $N=(D_N,F_N)$ de K et toute application ϕ_0 de X dans D_N, il existe un unique homomorphisme ϕ de M vers N qui prolonge ϕ_0 sur D_M.

○

Remarque 1.1 -- Si une algèbre K-libre sur X existe, elle est unique à un isomorphisme près, ce qui justifie le fait de parler de **la** F-algèbre K-libre engendrée par X.

Le théorème suivant dû à Birkhoff caractérise les classes d'algèbres pour lesquelles il existe une algèbre libre.

Théorème 1.1 -- [Birkhoff]-- Si K est une classe de F-algèbres stable par produit (vide aussi) et sous-algèbre, l'algèbre K-libre sur X existe pour tout ensemble X.

Dans le cas particulier où K est la classe de toutes les F-algèbres pour F fixé, et où soit X (l'ensemble des variables) soit F_0 l'ensemble des constantes est non vide, alors la F-algèbre libre engendrée par X existe. Elle est notée M(F,X).

Exemple 1.3 -- Ce théorème justifie donc l'existence des structures K-libres engendrées par un ensemble X, appelées structures libres comme groupe libre, monoide libre etc...

▽

Remarque 1.2 -- L'algèbre K-libre sur ø (unique à un isomorphisme prés) est l'objet initial de la classe des algèbres libres sur un ensemble X, lorsqu'on fait varier X, ce qui justifie l'appellation **d'algèbre initiale**. On le note **G(F)**. Il correspond à l'objet libre sur un ensemble vide de variables.

1.4. Les termes du premier ordre

Soient F un ensemble de symboles de fonction fixé, K la classe de toutes les F-algèbres. Soit **X** un ensemble d'éléments appelés **variables** et notés par les lettres x,y,z... Nous supposons toujours que soit F soit X est différent de l'ensemble vide. Alors le théorème précédent nous assure l'existence de l'algèbre K-libre sur X. Nous la notons **M(F,X)**.
Lorsque X = ø, M(F,ø) est l'algèbre initiale notée **G(F)**.

1.4.1. Termes: éléments de la F-algèbre K-libre

Définition 1.6 -- Nous appelons termes du premier ordre ou **termes** les éléments de M(F,X).

○

Définition 1.7 -- Les éléments de l'algèbre initiale sont appelés **termes clos** ou termes fermés.

○

Ces définitions des termes sont abstraites et ne permettent pas de les manipuler, nous allons donner dans le paragraphe suivant d'autres définitions explicites.

1.4.2. Termes : Arbres étiquetés

Soit $\mathbb{N}_+$ l'ensemble des entiers strictement positifs , $\mathbb{N}_+^*$ le monoide libre engendré et ε le mot vide et "." l'opération de concaténation.

Définition 1.8 -- Soit une application t de $\mathbb{N}_+^*$ dans F ∪ X et dom(t) son domaine de définition, c'est-à-dire l'ensemble des m appartenant à $\mathbb{N}_+^*$ tels que t(m) soit défini. Alors t est un **arbre** étiqueté sur F ∪ X si et seulement si

1) ε est élément de dom(t)

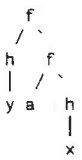
2) dom(t) est clos par préfixe i.e.: m appartient à dom(t) si m.m' appartient à dom(t).

3) pour tout m dans dom(t), m.i appartient à dom(t) si et seulement si i est compris entre 1 et l'arité de t(m).

Exemple 1.4 -- Soit F = {f, g, h, a} avec ar(f) = 2, ar(g) = ar(h) = 1 et ar(a) = 0 et V = {x,y}. L'application t définie sur {ε,1,2,11,21,22,221} par:

t(ε)=f , t(1)=h , t(2)=f , t(11)=y , t(21)=a , t(22)=h , t(221)=x

est un arbre que l'on représentera graphiquement ainsi:



Lemme 1.1 -- L'ensemble A des arbres sur F ∪ X est une F-algèbre, et ceci est réalisé en définissant pour tout f d'arité n dans F, l'opération f_A de A^n dans A :

$$f_A : t_1, t_2..,t_n \rightarrow \text{l'arbre } f(t_1,t_2,..,t_n)$$

Par abus de langage, le symbole de fonction f est confondu avec l'opération sur FUX f_A associée .

Proposition 1.1 -- La F-algèbre A du lemme précédent est K-libre sur X.

Comme toutes les F-algèbres K-libres sont isomorphes, M(F,X) et A l'ensemble des arbres sur FUX le sont également. Terme et Arbre correspondent donc à la même notion, nous parlons donc indifféremment des deux notions.

Exemple 1.5 -- L'arbre de l'exemple précédent correspond au terme f(h(y),f(a,h(x))).

a, x, f(x,y) sont des exemples de **termes**. a, b, f(a,b) sont des exemples de **termes clos** éléments de G(F).

∇

1.4.3. Terme : Définition inductive

Il est possible de construire l'ensemble des termes d'une façon inductive, cela permet de prouver dans M(F,X) des propriétés par induction sur la structure des termes.

F_k désigne l'ensemble des symboles de fonction d'arité k.

f_A l'opération associée au symbole f sur les arbres.

X l'ensemble des variables.

Soit $M_n(F,X)$ la famille de parties définies par:

-- $M_0(F,X) = X \cup \{a_A \mid a \text{ appartient à } F_0\}$

-- $M_{n+1}(F,X) = M_n(F,X) \cup \{f_A(t_1,\ldots,t_k)$ tel que f appartient à F_k et $t_1,\ldots,t_k$ appartiennent à $M_n(F,X)\}$

Lemme 1.2 -- M(F,X) est la réunion pour tous les n positifs ou nuls des $M_n(F,X)$.

Ce résultat montre le lien existant entre le raisonnement par récurrence structurelle et par récurrence sur les entiers. Une propriété de la forme "Pour tout t dans M(F,X), P(t)" peut être prouvée par récurrence sur la structure de t.

1.4.4. Opérations sur les termes

Notation -- L'ensemble des **occurrences** d'un terme t est le domaine de l'arbre t, on le note **O(t)**, c'est l'ensemble des **noeuds** de l'arbre désignés

par des mots de $\mathbb{N}_+^*$. ε est le mot vide et "." est l'opération de concatenation. Le mot m.i (ou tout simplement mi) désigne le i-ème fils du noeud m. $\bar{O}(t)$ est l'ensemble des occurrences non variables de t .

Un terme est **fini** si son domaine l'est.

Exemple 1.6 -- Soit le terme M = f(g(x),a) où x est une variable et a est une constante.

$$O(M) = \{\varepsilon,1,2,11\}$$
$$\bar{O}(M) = \{\varepsilon,1,2\}$$

∇

Définition 1.9 -- L'ensemble des variables d'un terme t, noté **V(t)** est l'ensemble des variables ayant une occurrence dans t, inductivement il est défini par :

-- $V(x) = \{x\}$ si $x \in X$

-- $V(f(t_1,t_2,..,t_n)) = \bigcup_{1\le i\le n} V(t_i)$

○

Définition 1.10 -- La **taille** d'un terme t, notée |t| est le cardinal de $\bar{O}(t)$, c'est donc le nombre des symboles non-variables dans t, il peut être défini inductivement par :

-- si t est une variable alors |t| = 0

-- si $t = f(t_1,\ldots,t_k)$ alors $|t| = 1 + \sum_{i=1}^{i=k} |t_i|$.

○

Exemple 1.7 -- Soit t = f(x,f(a,b)), |t| = 4, V(t) = {x}. Les éléments

de G(t) sont les éléments tels que V(t) = ø.

Définition 1.11 -- Soit t un terme et m un élément de O(t). **Le sous-terme de t à l'occurrence m**, noté $t_{|m}$ est l'arbre t' défini par t'(m')=t(m.m') pour tout m' appartenant à $\mathbb{N}_+^*$.

$$t_{|\varepsilon} = t$$
$$f(t_1,..,t_k)_{|im} = t_{i|m}$$

Définition 1.12 -- Nous notons $t_m[t']$, le terme t dans lequel le sous-terme à l'occurrence m a été remplacé par t'. Nous avons :

$$t_\varepsilon[t'] = t'$$
$$f(t_1,\ldots,t_k)_{im}[t'] = f(t_1,..,(t_i)_m[t'],..,t_k)$$

t[u] : u est un sous-terme de t

Définition 1.13 -- Pour tout terme u, $t^{-1}(u)$ est l'ensemble des occurrences de u dans t, noté **O(u,t)**. Si u n'est pas un sous-terme de t, O(u,t) = ø. Le cardinal de O(u,t) est noté **#(u,t)**, c'est le nombre d'occurrences de u dans t.

Définition 1.14 -- Un terme t est dit **linéaire** si toute variable a au plus une occurrence dans t, c'est-à-dire si

$$\forall x \in V,\ \#(x,t) \leq 1.$$

1.5. Endomorphismes de M(F,X)

Définition 1.15 -- Une **substitution** σ est une application de X dans M(F,X) telle que σ(x) = x sauf sur un sous-ensemble fini appelé **domaine** de σ et noté D(σ). Lorsqu'il n'y a pas d'ambiguité, σ(x) est notée σx.

$$\mathbf{D(\sigma)} = \{x \mid \sigma x \neq x\}$$

I(σ) est l'ensemble des variables introduites défini par :

$$\mathbf{I(\sigma)} = \bigcup_{x \in D(\sigma)} V(\sigma x)$$

Le caractère libre de la F-algèbre M(F,X), nous permet de prolonger σ de façon unique en un endomorphisme de M(F,X). Il vérifie donc pour tout f d'arité k de F, pour tout $t_1,\ldots,t_k$ de M(F,X) :

$$\sigma f(t_1,\ldots,t_k) = f(\sigma t_1,\ldots,\sigma t_k).$$

Notation -- Les substitutions sont notées par les lettres α,β,ξ... et représentées par leurs graphes notés:

$\{(x \leftarrow \sigma x)\}$ pour x dans le domaine de σ.

Définition 1.16 -- La composition des substitutions est la composition classique des applications :

$$\forall x \in X \quad (\sigma\rho)x = \sigma(\rho x)$$

O

Définition 1.17 -- Pour toute substitution σ et pour tout sous-ensemble U de X, la restriction de σ à U notée $\sigma_{|U}$ est la substitution définie par :

$$\sigma_{|U}(x) = \sigma(x) \text{ si } x \in U$$
$$= x \text{ sinon}$$

O

Définition 1.18 -- Soient M et N deux termes. On apelle problème de **filtrage** de M vers N, le problème qui consiste à trouver toutes les substitutions σ telles que $\sigma M = N$. L'**unification** de M et N est le problème de trouver toutes les substitutions σ telles que $\sigma M = \sigma N$.

O

On note **Subs** l'ensemble de toutes les substitutions.

1.6. Préordre de Filtrage

La composition des substitutions induit un préordre sur l'ensemble Subs.

Définition 1.19 -- Nous dirons que σ est **inférieur** à ρ, ce qu'on note $\sigma \leq \rho$ si et seulement s'il existe η tel que $\eta\sigma = \rho$.

O

La relation d'équivalence $\equiv$ associée à ce préordre est l'égalité des substitutions à une permutation près. $\sigma \equiv \rho$ ssi $\sigma \leq \rho$ et $\rho \leq \sigma$.

Les substitutions induisent aussi un ordre dit de filtrage sur les termes.

Définition 1.20 -- M est dit **inférieur** à N si et seulement s'il existe σ tel que $\sigma M = N$. On dit aussi que N est une **instance** de M, ou que M **filtre** N ou qu'il existe un filtre de M vers N.

Unifier t_1 et t_2 c'est chercher les instances communes de deux termes t_1 et t_2

O

Les deux termes M et N sont **unifiables** si et seulement s'il existe une substitution σ telle que $\sigma M = \sigma N$.

O

Nous dirons que σ est un filtre (resp. unificateur) **principal** si et seulement si σ est inférieur à toutes les substitutions ρ telle que $\rho M = N$ (resp. $\rho M = \rho N$). On peut voir que les unificateurs principaux sont uniques à une permutation des variables près, c'est-à-dire modulo $\equiv$.

L'algorithme de filtrage est simple et peut être donné par une procédure recursive [Huet76].

Pour le cas de l'unification le problème est moins facile. Depuis les travaux de Herbrand [Herbrand] en 1930, l'étude de l'unification a donné lieu à de nombreux travaux. Robinson [Robinson-71] est le premier à donner un algorithme d'unification. Mais il a été très vite l'objet de critique à cause de la complexité exponentielle de son algorithme sur la structure de

terme [Huet-76,Baxter-73,Martelli-Montanari-82,Corbin-Bidoit-83]. Des études expérimentales [Hullot80] ont montré la superiorité en pratique de l'algorithme de Martelli et Montanari pour une structure particulière des termes. C'est la généralisation de cet algorithme que C. Kirchner a étudié pour construire des algorithmes d'unification dans les théories équationnelles.

2. Variétés et Théories équationnelles

2. Variétés et théories équationnelles

2.1. Variétés équationnelles

Définition 2.1 -- Une **équation** est une paire de termes $\{t_1,t_2\}$ notée $t_1 = t_2$. Les équations peuvent être considérées comme une expression dans une algèbre particulière, avec "=" comme symbole de fonction.

○

Exemple 2.1 -- F={.},

$$x.y = y.x$$

est une équation.

▽

Définition 2.2 -- On dit q'une F-algèbre $M=(D_M,F_M)$ **valide** l'équation $(t_1 = t_2)$ ou que M est un **modèle** de $t_1=t_2$ si et seulement si pour tout homomorphisme ϕ de M(F,X) dans D_M, $\phi(t_1) = \phi(t_2)$. On dit aussi que l'équation $(t_1 = t_2)$ est valide dans M et on le note

$$M \vdash (t_1=t_2).$$

Rappelons qu'un homomorphisme de l'algèbre des termes M(F,X) dans l'algèbre M est complètement déterminé par sa restriction ϕ_0 sur l'ensemble des variables X. Comme de plus, nous ne nous intéressons qu'aux variables

de t_1 et t_2, il suffit de se donner ϕ_0 sur $V(t_1) \cup V(t_2)$.

○

Exemple 2.2 -- L'agèbre $(\mathbb{B},\{|_{\mathbb{B}},\&_{\mathbb{B}}\})$ de l'exemple 1.1 valide l'équation

$$x \mid x = x$$

▽

Définition 2.3 -- Une **variété équationnelle** de F-algèbre est une classe de F-algèbres H pour laquelle il existe un ensemble d'équations A tel que H soit la classe de toutes les algèbres validant les équations de A. H est aussi appelée la **classe des modèles de A** et est notée **H(A)**.

○

On sait (Birkhoff) que, pour tout X, la F-algèbre M(A)-libre engendrée par X existe, dès que M(A) est une variété.

Le problème qui consiste à décider si une équation $(t_1 = t_2)$ est valide dans toute F-algèbre de M(A) est appelé **problème du mot** pour la F-algèbre M(A)-libre. Le problème de la validité dans M(A) revient à caractériser les formules de la théorie équationnelle de A c'est-à-dire les équations valides dans toutes les algèbres de M(A).

2.2. Congruence

Définition 2.4 -- Soit $M=(D_M,F_M)$ une F-algèbre. Une relation d'équivalence $\equiv$ sur D_M est une F-congruence (ou **congruence**) sur M si et seulement si pour tout symbole de fonction f de F d'arité n et pour tout

élément $t_1,\ldots,t_n,t'_1,\ldots,t'_n$ dans D_M :

$$t_1 \equiv t'_1 ,\ldots, t_n \equiv t'_n \Rightarrow f(t_1,\ldots,t_n) \equiv f(t'_1,\ldots,t'_n).$$

○

Définition 2.5 -- Soit $M=(D_M,F_M)$ une F-algèbre et $\equiv$ une congruence sur M. **L'algèbre quotient** de M par $\equiv$ est le couple $M=(D_M/\equiv,F_M/\equiv)$ où $D_M/\equiv$ est l'ensemble quotient de D_M par $\equiv$, et $F_M/\equiv$ est l'ensemble des opérations induites dans le quotient.

○

Définition 2.6 -- Soit A un ensemble d'équations. La plus petite congruence sur M(F,V) contenant toutes les paires $\{\sigma(t_1), \sigma(t_2)\}$, où $(t_1 = t_2)$ est une équation quelconque de A et σ une substitution, est appelée **A-égalité** ou **congruence engendrée par A** ou **égalité modulo A** et est notée $=_A$. Deux termes tels que $t_1 =_A t_2$ sont dit **A-égaux**.

○

Une manière équivalente de définir l'égalité modulo A est de définir $=_A$ comme la fermeture réflexive symétrique et transitive de la relation $|-|_A$ appelée **remplacement en une étape** et définie pour M et N deux termes par : $M \,|-|_A\, N$ si et seulement s'il existe l=r équation dans A, une occurrence u dans $O(M)\cap O(M)$ et une substitution σ telle que $M_{|u}=\sigma(l)$ et $N_{|u}=\sigma(r)$, ou $M_{|u}=\sigma(r)$ et $N_{|u}=\sigma(l)$.

2.3. Théorie équationnelle

Définition 2.7 -- On appelle **théorie équationnelle engendrée par A** l'algèbre quotient $M' = (M(F,X)/=_A,F)$, on dit aussi théorie équationnelle presentée par A ou tout simplement **théorie équationnelle A**. A est alors une **présentation** de M'. Lorsque A = ø on parle de **théorie équationnelle vide** qui correspond à M(F,X).

○

Dans toute la suite de ce travail, on suppose que les congruences à considérer sont les égalités engendrées par un nombre fini d'équations, appelées aussi **axiomes**. Le théorème de Birkhoff assure l'existence de l'algèbre libre sous ces hypothèses. Ainsi nous parlerons de groupe libre. Mais la théorie des corps n'est pas spécifiable par un ensemble fini d'équations, de plus elle n'est pas fermée par produit (voir theorème 1.1 de Birkhoff). Il n'existe donc pas de corps libre. Henkin [Henkin-84] Donne aussi une théorie des entiers naturels qui n'est pas finiment spécifiable.
Nous supposons aussi qu'aucune présentation de la théorie ne contient un axiome de la forme x = y avec x et y ∈ X. Cela évitera à la théorie équationnelle d'être réduite à un seul élément (on parle d'inconsistance).

Ainsi nous parlerons de théorie équationnelle associative commutative c'est-à-dire de la théorie équationnelle engendrée par l'ensemble des axiomes suivant :

A = {associativité, commutativité}

où l'axiome d'associativité est x.(y.z) = (x.y).z et celui de commutativité est x.y = y.x.
x, y et z étant des variables dans X et "." un élément de F.

Définition 2.8 -- Une théorie équationnelle A est dite **régulière** si et seulement si il existe une représentation de A dont tous les axiomes $t_1=t_2$ satisfont $V(t_1)=V(t_2)$; elle est dite **finie** si et seulement si les classes d'équivalences modulo $=_A$ contiennent un ensemble fini d'éléments, et elle est dite **potente** si et seulement si il existe une représentation de A qui contient un axiome du type x=t ou x ∈ X et t ∉ X. Une théorie est dite **permutative** si et seulement si pour A une présentation, pour tous les axiomes l=r de A et pour tous les termes (ou variables) t, on a #(t,l) = #(t,r).

○

Exemple 2.3 -- La théorie distributive est finie la démonstration se trouve dans [Szabo-82]. La théorie réduite à l'axiome d'idempotence est potente. La théorie commutative est permutative. La théorie constituée d'un ensemble fini de symbole associatif-commutatif est permutative. les théories permutatives sont finies [Burckert-Herold-86]. La théorie réduite au seul axiome f(x,y)=x n'est pas regulière.

▽

2.4. Théorème de Complétude

L'algèbre quotient de M(F,X) par $=_A$ est l'algèbre M(A)-libre sur X, le raisonnement équationnel est suffisant pour décider l'égalité dans une théorie équationnelle comme le montre le théorème suivant de Birkhoff.

Théorème 2.1 -- [Birkhoff]-- Etant donné un ensemble d'équations A, une équation $(t_1 = t_2)$ est valide dans la variété M(A) si et seulement si

$t_1 =_A t_2$.

$$M(A) \vdash (t_1 = t_2) \Leftrightarrow t_1 =_A t_2$$

Ce théorème est le fondement de la logique équationnelle. il permet de ramener un problème sémantique de validité d'une équation dans une variété équationnelle M(A), à un problème purement syntaxique de A-égalité.

2.5. Algèbre initiale de la théorie

L'algèbre quotient de G(F) (qui est rappelons-le M(F,ø)) par $=_A$ est l'objet initial de la catégorie M(A) [Goguen Thatcher Wagner], il est appelé **algèbre initiale de la théorie** et noté **G(F,A)**.

En appliquant la définition de validité 1.22 nous obtenons :

Définition 2.9 -- Une équation est valide dans l'algèbre initiale G(F,A) si et seulement si pour toute substitution σ close (i.e X→G), nous avons $\sigma(t) =_A \sigma(t')$.

○

Sauf pour le problème d'égalité des termes clos, le raisonnement équationnel n'est pas complet pour le problème des mots dans l'algèbre initiale en général, car il est possible d'avoir deux termes contenant des variables et non égaux modulo E, mais tels que toute substitution close les rende E-égaux. D'autres règles d'inférence, comme la reccurrence structurelle, sont alors nécessaires [Goguen-80,Huet-Hullot-82,Remy-82,Paul-

84,Kirchner-85b,Jouannaud-Kounalis-86]... Par exemple l'équation

$$rev(rev(x)) = x$$

ne peut pas être prouvée dans la spécification suivante:

```
null + x = x
(x.y) + z = x.(y + z)
rev(null) = null
rev(x.y) = rev(y) + (x.null)
```

LA preuve ne peut pas se faire uniquement par remplacement d'égaux par des égaux, mais elle nécessite un raisonnement par reccurrence.

2.6. E-filtrage et E-unification

La E-égalité peut être étendue pour les substitutions.

Définition 2.10 -- Nous disons que deux substitutions σ et ρ sont E-égales sur un sous-ensemble V de X et nous notons $\sigma =_E \rho\ [V]$ si et seulement si pour tout $x \in V$ $\sigma x =_E \rho x$.

○

Définition 2.11 -- σ est dites **inférieure** à ρ sur V dans la théorie E ce qu'on note $\sigma \leq_E \rho\ [V]$ si et seulement s'il existe une substitution η telle que $\eta\sigma =_E \rho\ [V]$.

○

Définition 2.12 -- $\sigma \equiv_E \rho\ [V]$ ssi $\sigma \leq_E \rho\ [V]$ et $\rho \leq_E \sigma\ [V]$.

○

Définition 2.13 -- σ est un **E-filtre** du terme M vers le terme N ssi $\sigma M =_E N$. σ est aussi appelé filtre de M vers N dans la théorie E.

L'ensemble des E-filtres de M vers N est noté $\{M \ll N\}_E$.

$$\{M \ll N\}_E = \{\sigma \ / \ \sigma M =_E N\}$$

○

Nous avons un préordre dit de E-filtrage sur les termes et noté aussi $\leq_E$: $M \leq_E N$ ssi il existe σ telle que $\sigma M =_E N$.

Définition 2.14 -- Deux termes M et N sont dits **E-unifiables** ssi il existe une substitution σ telle que $\sigma M =_E \sigma N$. σ est dit **E-unificateur** de M et N, ou unificateur dans la théorie E. L'ensemble des E-unificateurs de M et N est noté $\{M =? N\}_E$.

$$\{M =? N\}_E = \{\sigma \ / \ \sigma M =_E \sigma N\}$$

○

Les filtres ne sont pas des unificateurs particuliers, l'exemple classique est que x et f(x) sont filtrables, sans qu'ils ne soient unifiables. Mais si N ne contient pas de variable ou si V(N) et V(M) sont disjoints, l'unification et le filtrage de M et N sont équivalents.

Contrairement au cas où E = ø, l'ensemble des filtres (resp. des unificateurs) principaux n'est plus réduit à un seul élément, il peut même être infini. Pire encore, pour certaines théories le problème de l'unification est indécidable comme par exemple la théorie Associative-Distributive (AD) [Szabo82] [Arborg85]. Pour les théories semi-décidables E, $\{M =? N\}_E$ est récursivement énumérable car l'ensemble de toutes les substitutions est lui-même énumérable. Plotkin [Plotkin-72] a défini un ensemble générateur de l'ensemble de tous les unificateurs de M et N, appelé

ensemble complet de E-unificateurs et noté $C\{M =? N\}_E$. Cet ensemble permet d'engendrer tous les E-unificateurs par instantiation:

$$\forall \sigma \in \{M =? N\}_E, \forall \rho \in Subs, \ \rho\sigma \in \{M =? N\}_E$$

Un ensemble d'unificateurs $C\{M =? N\}_E$ est générateur si

$$\forall \sigma \in \{M =? N\}_E, \exists \rho \in c\{M =? N\}_E \ / \ \rho \leq_E \sigma$$

Cet ensemble est une base s'il vérifie la condition de minimalité:

$$\forall \sigma \text{ et } \rho \in c\{M =? N\}_E, \ \sigma \leq_E \rho \text{ implique } \sigma = \rho$$

Plus précisement nous avons la définition suivante:

Définition 2.15 -- Soient M et N deux termes, $V = V(M) \cup V(N)$ et W un ensemble fini de variables contenant V. Un ensemble **complet de E-unificateurs** de M et N en dehors de W, noté $c\{M =? N\}_E$, est un ensemble de substitutions vérifiant:

1) $\forall \sigma \in c\{M =? N\}_E$, $D(\sigma) \subseteq V$ et $I(\sigma) \cap W = \emptyset$

2) $c\{M =? N\}_E \subseteq \{M =? N\}_E$

3) $\forall \sigma \in \{M =? N\}_E, \exists \rho \in c\{M =? N\}_E \ / \ \rho \leq_E \sigma \ [V]$

Nous dirons que $c\{M =? N\}_E$ est minimal, et on le note $\mu c\{M =? N\}_E$, si de plus

4) $\forall \sigma \text{ et } \rho \in c\{M =? N\}_E, \ \sigma \leq_E \rho \text{ implique } \sigma = \rho$

○

La condition 1) est une condition technique qui permet d'éviter les conflits entre les variables des termes M et N et les variables introduites

par σ. La condition 2) traduit la cohérence et la condition 3) la complétude. La condition 4) décrit l'indépendance des unificateurs.

De la même manière on définit un ensemble complet de E-filtres.

Définition 2.16 -- Soient M et N deux termes, V = V(M) et W un ensemble fini de variables contenant V. Un ensemble **complet de E-filtres** de M vers N en dehors de W, noté $c\{M\ll N\}_E$, est un ensemble de substitutions vérifiant:

1) $\forall\ \sigma \in c\{M\ll N\}_E$, $D(\sigma) \subseteq V$ et $I(\sigma) \cap (W\text{-}V(N)) = \emptyset$

2) $c\{M\ll N\}_E \subseteq \{M\ll N\}_E$

3) $\forall\ \sigma \in \{M\ll N\}_E$, $\exists\ \rho \in c\{M\ll N\}_E$ / $\rho \leq_E \sigma\ [V]$

Nous dirons que $c\{M\ll N\}_E$ est **minimal** , et on l'écrit $\mu c\{M\ll N\}_E$, si de plus

4) $\forall\ \sigma$ et $\rho \in c\{M\ll N\}_E$, $\sigma \leq_E \rho$ implique $\sigma = \rho$

▷ Notons que l'ensemble minimal de E-unificateurs (rep. E-filtres) n'existe pas toujours [Fages-Huet-83a]. L'exemple donné par Fages et Huet n'est pas évident, il montre que la minimalité 4) peut être incompatible avec la complétude 3), à cause de chaines infinies non bornées d'unificateurs de plus en plus généraux. Mais si un ensemble minimal existe, il est unique modulo $=_E$ (unicité de la base).

▷ L'ensemble complet de E-unificateurs (resp. E-filtres) existe toujours (lorsque l'unification est décidable bien sûr), il suffit de considérer tous les unificateurs (resp. filtres) satisfaisants 1),

puisque W est fini et V infini dénombrable.

▷ Contrairement à ce qu'on pourrait croire, le filtrage n'est pas un cas particulier de l'unification, pour une équation de filtrage $M\ll_E N$ dans une théorie E, il ne suffit pas de renomer les variables de N par des nouvelles constantes, et appliquer un algorithme d'E-unification, en effet, Burckert [Burckert86] a montré qu'en ajoutant des nouvelles constantes à la théorie E, la E-unification peut devenir indécidable.

▷ L'ensemble complet de E-unificateurs (resp. E-filtres) peut ne pas être fini, par exemple [Plotn 72] l'ensemble des solutions de l'équation a+x = x+a losque + est associative est $\{x\leftarrow a, x\leftarrow a+a, x\leftarrow a+a+a, \ldots\}$. Il est infini et tous ses éléments sont incomparables. Pour le cas du filtrage, un exemple est construit par F. Fages [Fages-83] où l'ensemble minimal de E-filtres est infini, donc l'ensemble complet est lui même infini.

▷ Quand il existe un ensemble complet d'unificateurs (resp. de filtres) fini, il existe toujours un ensemble minimal, obtenu en éliminant les éléments redondants.

▷ Filtrage et unification ne se comportent pas de la même manière dans les théories régulières (théories définies par des équations $t_1 = t_2$ telles que $V(t_1) = V(t_2)$). Dans de telles théories, alors que pour le filtrage, tout ensemble complet de E-filtres distincts (modulo $=_E$) est minimal, pour l'unification Fages [Fages83] a montré qu'il peut exister des termes E-unifiables qui n'admettent pas d'ensemble complet de E-unificateurs minimal.

- Burckert [Burckert-86] a étudié les relations qui peuvent exister entre la E-unification et le E-filtrage en utilisant une notion plus générale que le filtrage appelée **E-unification restreinte**, qui est l'unification sur un ensemble fixé de variables. Il a montré que dans le cas des théories non potentes, l'ensemble minimal des unificateurs restreints est aussi un ensemble minimal d'unificateurs.

- Remarquons qu'il existe des procédures d'unification qui énumèrent l'ensemble de tous les unificateurs, des procédures qui énumèrent un ensemble complet d'unificateurs et des procédures qui énumèrent un ensemble complet et minimal de solutions. Ces dernières procédures sont appelées des **procédures d'unification minimales.**

L'état de l'art en 85 sur l'unification équationnelle se trouve dans [Kirchner85]. Il s'agit, en fait, d'un domaine de recherche en pleine expansion.

Nous revenons au problème du filtrage équationnel dans le Partie II de cette thèse où nous présentons une méthode similaire à la méthode utilisée dans [Kirchner85] pour construire des algorithmes de filtrage.

3. Systèmes de réécriture

3. Systèmes de réécriture

3.1. Introduction

Nous avons vu que le théorème de Birkhoff est un résultat fondamental puisqu'il ramène un problème sémantique (la validité d'une équation dans une variété équationnelle : problème du mot) à un problème purement syntaxique de A-égalité ($=_A$). Par exemple pour décider de la validité de l'équation 0+x=x dans la variété d'algèbres définie par les trois équations suivantes :

$$x+0 = x$$
$$x+(-x) = 0$$
$$(x+y)+z = x+(y+z)$$

Il suffit d'avoir $1+x =_A x$.

Bien que ce résultat donne une solution mathématique pour le problème du mot, il n'est pas suffisant pour une solution informatique implantable sur machine. En effet, le raisonnement équationnel ($=_A$) utilisé par le mathématicien, c'est-à-dire le remplacement d'égaux par égaux, n'est pas utilisable en machine à cause des nombreux retours en arrière nécessaires, ce qui peut engendrer des boucles incontrôlables et de ce fait rendre la méthode inefficace.

La réécriture est une approche développée pour éviter cet

inconvénient. Il s'agit d'orienter les équations, c'est-à-dire de ne les utiliser que dans un seul sens. Un sens d'orientation des équations de l'exemple est le suivant:

$$x+0 \rightarrow x$$
$$x+(-x) \rightarrow 0$$
$$(x+y)+z \rightarrow x+(y+z)$$

Les équations ainsi orientées sont appelées règles de réécriture. Réécrire un terme t consiste à choisir un sous-terme t' de t et une règle de réécriture $g \rightarrow d$ tel que $\sigma(g)=t'$ (i.e. g filtre t' ou bien t' est une instance de g) puis à remplacer t' dans t par l'instance correspondante dans d (σd). Ainsi $(x+y)+(-y) \rightarrow x+(y+(-y)) \rightarrow x+0 \rightarrow x$, en utilisant dans l'ordre la troisième, la deuxième et la première règle . Un terme qui ne peut être réécrit par aucune règle de réécriture du système est dit irréductible ou en forme normale.

Du fait de la réstriction de l'utilisation des équations dans un seul sens, le système de réécriture n'a plus la même puissance de démonstration que l'ensemble des équations E. La procédure de complétion de Knuth-Bendix est une méthode qui à partir d'un ensemble d'équations E, donne, si elle termine, un système de réécriture qui a la même puissance sur les termes que l'ensemble d'équations E; ce qui permet de décider de la A-égalité à l'aide de la réécriture uniquement, à condition d'avoir la propriété de terminaison (qui implique l'existence d'une forme normale).

L'unicité de la forme normale est forcée dans l'algorithme de Knuth-Bendix par l'adjonction des 2 formes normales différentes, si elles existent, comme nouvelle équation à orienter. Cela nous assure la propriété

appelée confluence.

Un système qui termine et a la proprieté de confluence est appelé système convergent et donne une procédure de décision pour le problème du mot en comparant les forme normales des deux termes de l'équation à tester. Par exemple le système précédent de trois règles est complété par la procédure de Knuth-Bendix en le système convergent suivant:

$$-(0) \rightarrow 0$$
$$0 + x \rightarrow x$$
$$x + 0 \rightarrow x$$
$$-(-(y)) \rightarrow y$$
$$-(x) + x \rightarrow 0$$
$$y + -(y) \rightarrow 0$$
$$-(x) + (x + y) \rightarrow y$$
$$x + (-(x) + y) \rightarrow y$$
$$(x + y) + z \rightarrow x + (y + z)$$
$$-(x + y) \rightarrow -(y) + -(x)$$

Dans certains cas, il n'est pas possible d'orienter une équation de A sans perdre la propriété de terminaison, comme par exemple pour le cas de l'équation de commutativité $x+y=y+x$ où il existe toujours une chaine infinie pour la relation $\rightarrow_C$ qui est $x+y \rightarrow y+x \rightarrow x+y \ldots$ C'est le cas aussi des axiomes d'associativité et de commutativité $x+(y+z) \rightarrow (y+z)+x \rightarrow y+(z+x) \rightarrow \ldots$

Pour résoudre ce problème une idée est alors de diviser l'ensemble des axiomes A en deux ensembles R et E, et de considérer la réécriture des classes qui est alors la composition des trois relations $(=_E)(\rightarrow)(=_E)$, c'est ce qu'on appelle la réécriture équationnelle. La réécriture classique devient alors un cas particulier de la réécriture équationnelle, c'est le cas où l'ensemble des équations E est vide.

En fait, il est nécessaire de montrer que la réécriture de n'importe quel terme de la classe ne dépend pas du terme choisi dans cette classe, comme cela se fait quand on calcule sur les classes d'équivalence. Après plusieurs tentatives d'élaboration de nouveaux concepts abstraits permettant la manipulation conjointe de règles et d'équations [Sethi-74,Huet-81,Peterson-Stickel-81,Jouaaud-83], J-P. Jouannaud et H. Kirchner [Jouannaud-Kirchner-84] ont mis en évidence le concept de cohérence qui a permis de faire un parallèle entre la réécriture équationnelle et la réécriture standard. Ils ont aussi donné un cadre général permettant d'englober différentes méthodes de réécriture équationnelle. Nous présentons au chapitre suivant une méthode de réécriture manipulant des règles et des équations, différente de la réécriture équationnelle.

Nous avons choisi d'utiliser un formalisme proche de celui de Claude et Hélène Kirchner [Kirchner-85,Kirchner-85ba], qui généralise celui de Jouannaud [Jouannaud-83] par ce qu'il permet d'englober les différentes définitions et de faire la synthèse des travaux existants dans ce domaine.

3.2. Notions de base

Nous avons déjà défini les notions de filtrage et d'unification (paragraphe 2.6). Nous utilisons ici un autre formalisme: on définit le filtrage et l'unification comme des applications U_E (unification particulière) et F_E (filtrage particulier) qui à un couple de termes font correspondre un ensemble, éventuellement vide, de substitutions. L'intérêt de ce point de vue est de permettre ensuite de faire varier la méthode de

filtrage ou d'unification en fonction des termes considérés et de regrouper les différentes méthodes de réécriture dans le même formalisme.

Rappelons que $M(F,X)$ est l'algèbre des termes et Subs est l'ensemble des substitutions.

Définition 3.1 -- Soient M et N des termes de $M(F,X)$. Une opération de filtrage est une application notée F_E de $M(F,X)\times M(F,X)$ dans $P(Subs)$.

○

$F_E(M,N)$ est par exemple l'unique substitution σ telle que $\sigma M = N$, dans ce cas F_E est le filtrage dans la théorie vide et $F_E(M,N)=\{M\ll N\}_\emptyset$. Un autre exemple pour F_E: l'application qui aux termes M et N fait correspondre $\{M\ll N\}_E$ (def.2.13), elle est naturellement notée $\{\ll\}_E$.

Il est possible de considérer d'autres types de filtrage moins intuitifs comme par exemple $F_E(M,N)$ l'application définie comme étant $\{\ll\}_E$ si M est linéaire et $\{\ll\}_E$ sinon.

Définition 3.2 -- Une équation orientée $s\rightarrow t$ telle que $V(t) \subseteq V(s)$ est appelée **règle de réécriture**.

○

Définition 3.3 -- Soit A une théorie et soient E et R deux ensembles d'équations tels que A=EUR. les équations g=d de R sont orientées en règles de réécriture $g\rightarrow d$. La **relation de réécriture** $\rightarrow_{RE}$ associée à l'opération de filtrage F_E et à R est définie comme suit: $s\rightarrow_{RE} t$ si et seulement si

1) il existe une règle $g\rightarrow d$ de R et une occurrence u de s telles que $\sigma \in F_E(g,s_{|u})$ et,

2) $t = s_u[\sigma d]$

A chaque type de filtrage correspond un type de réécriture.

- Remarquons tout de suite que lorsque que nous considérons $F_E=\{«\}_E$ avec $E=\emptyset$ (et donc $F_E=\{«\}_\emptyset$), la relation de réécriture $\rightarrow_{R\emptyset}$ notée $\rightarrow_R$ est la suivante: $s\rightarrow_R t$ si et seulement s'il existe une règle $g\rightarrow d$ de R telle que $s=u[\sigma g]$ et $t=u[\sigma d]$.

- La réécriture de Peterson et Stickel notée $\rightarrow_{R,E}$ est associée à $\{«\}_E$.

Notation -- Nous notons $\rightarrow_{R/E}$ la composition $=_E\rightarrow_R=\rightarrow$.

Nous avons

$$\rightarrow_R \subseteq \rightarrow_{RE} \subseteq \rightarrow_{R,E} \subseteq \rightarrow_{R/E}$$

Définition 3.4 -- Un terme t est dit **RE-irréductible** s'il n'existe aucun terme u tel que $t\rightarrow_{RE}u$. Un terme u est une **RE-forme normale** de t si $t\rightarrow^*_{RE}u$ et u est RE-irréductible.

La relation $\rightarrow_{RE}$ est **noethérienne modulo E** si et seulement si $\rightarrow_{R/E}$ est noethérienne.

Une paire de termes (t_1,t_2) est dite:

	notée	
R/E-confluente	$t_1\downarrow_{R/E}t_2$	il existe t tel que $t_1\rightarrow^*_{R/E}t$ et $t_2\rightarrow^*_{R/E}t$
RE-confluente modulo E	$t_1\downarrow\downarrow t_2$	il existe t_1' et $t'2$ tels que $t_1\rightarrow^*_{R/E}t_1'$ et $t_2\rightarrow^*_{R/E}t_2'$ et $t'1=_E t'2$

La relation $\rightarrow_{RE}$ est dite:

Church Rosser modulo E	pour tous termes t_1 t_2 $t_1 =_A t_2 \Rightarrow t_1 \downarrow\downarrow t_2$
Confluente modulo E	pour tous termes t t_1 t_2 $t \rightarrow^*_{RE} t_1$ et $t \rightarrow^*_{RE} t_2 \Rightarrow t_1 \downarrow\downarrow t_2$
Coherente modulo E	pour tous termes t t_1 t_2 $t \rightarrow^*_{RE} t_1$ et $t =_E t_2 \Rightarrow t_1 \downarrow\downarrow t_2 \Rightarrow$ il existe t_2' tel que $t_2 \rightarrow_{RE} t_2'$ et $t_1 \downarrow\downarrow t_2'$
compatible avec le préodre $=<^\sigma$ associée à F_E sur t et t'	$t =<_\sigma t'$ et t est RE-irréductible en $t_1 \Rightarrow$ t' est RE-irréductible en t'1 et t'1 $\downarrow_{RE}$ $\sigma(t_1)$.
compatible au sommet avec le préordre $=<^\sigma$ associé à F_E sur t et'	$t =<_\sigma t'$ et t est RE-irréductible en t_1 à l'occurrence $\varepsilon \Rightarrow$ t' est RE-irréductible en t'1 et t'1 $\downarrow_{RE}$ $\sigma(t_1)$.

○

Nous avons le résultat fondamental suivant:

Théorème 3.1 --[Jouannaud-Kirchner-86]-- (Théorème de Church-Rosser modulo)

Si $\rightarrow_{RE}$ est noethérienne, les propriétés suivantes sont équivalentes:

(1) $\rightarrow_{RE}$ est de Church-Rosser modulo E

(2) $\rightarrow_{RE}$ est confluente modulo E et cohérente modulo E

(3) pour tous termes t et t', $t =_A t'$ si et seulement si $t\downarrow_{RE} =_E t'\downarrow_{RE}$.

De la même manière que le filtrage nous définissons l'unification comme une application U_E.

Définition 3.5 -- Soient M et N deux termes de M(F,X). Une opération de d'unification U_E est une application de M(F,X)xM(F,X) dans P(subs).

○

$U_E(M,N)$ est par exemple l'unique unificateur σ tel que $\sigma M = \sigma N$, σ est dans ce cas l'unificateur principal dans la théorie vide.

A toute notion d'unification il est associé une notion de paires critiques que nous définissons ici:

Définition 3.6 -- Un terme t non réduit à une variable se U_E-superpose sur un terme t' à une occurrence u de $\bar{O}(t')$ avec un ensemble complet T de superpositions si et seulement si $T = U_E(t, t'|_u)$.

○

Exemple 3.1 -- Soit les deux termes t=f(x,a) et t'=g(f(b,y),b). t se U_σ-superpose à l'occurrence 1 $T=\{x \leftarrow b, y \leftarrow a\}$.

▽

Définition 3.7 -- Etant données deux règles $g \rightarrow d$ et $l \rightarrow r$ telles que V(g) et V(l) soient disjoints, et que l se superpose sur g à l'occurrence u avec l'ensemble complet de superpositions T, alors l'ensemble

$\{(p,q) \mid p=\tau(d),\ q=\tau(g[u\leftarrow r])$ pour tout τ dans $T\}$

est un **ensemble complet de paires critiques** de la règle $l\rightarrow r$ dans la règle $g\rightarrow d$ à l'occurrence u. A chaque superposition τ de T est associé le terme superposé $\tau(g)$.

□

- On retrouve la définition de paires critique définie par Knuth et Bendix en prenant $U_E = U_\emptyset$.
- On retrouve la notion de E-pair critique définie par Peterson et Stickel en prenant U_E l'ensemble complet des E-unificateurs de tous les termes.

Dans ce formalisme le théorème de décidabilité de la propriété de Church-Rosser s'énonce ainsi:

Théorème 3.2 -- Soit R un ensemble de règles E-noethérien, et E une théorie telle que U_E existe et que les classes d'équivalence modulo $=_E$ soient finies. Supposons que $\rightarrow_{RE}$ soit la réunion d'un nombre fini n de relations de réécriture équationnelles $\rightarrow_{R_iE}$ telles que

- pour tout $i \in [1..k]$, $\rightarrow_{R_{iE}}$ est compatible sur $M(F,X)\times M(F,X)$ avec l'ordre associé $=<^i$ qui est conservé sur les sous-termes,
- pour tout $i \in [k+1..n]$, $\rightarrow_{R_iE}$ est compatible au sommet avec l'ordre associé $=<^i$ sur $M(F,X)\times M(F,X)$.

Alors $\rightarrow_{RE}$ est Church-Rosser modulo E si et seulement si

- toutes les paires critiques de confluence sont R/E-confluentes modulo E
- toutes les paires critiques de cohérence (p,q) obtenues par superposition d'une règle dans un axiome sont telles qu'il existe p' tel que $p \rightarrow_{RE} p'$ et (p',q) soit R/E-confluente.
- toutes les paires critiques de cohérence (p,q) obtenues par superposition d'un axiome dans une règle de $\bigcup_{i=1}^{k} R_i$ sont telles qu'il existe q' tel que $q \rightarrow_{RE} q'$ et (p,q') soit R/E-confluente.

J-P. Jouannaud et H. Kirchner ont donné une procédure de complétion équationnelle lorsque U_E existe, elle prend en argument un ensemble P de paires de termes, un ensemble R de règles de réécriture, un ensemble constant d'axiomes E et un ordre compatible avec la structure de F-algèbre tel que l'équivalence associée contiennent $=_E$ et tel que l'ordre strict associé soit noethérien. Lorsque U_E existe et lorsque E est une théorie finie et que le préordre de E-filtrage est nothérien, alors $1 \rightarrow_{R_\infty E}$ est Church-Rosser modulo E, R_∞ étant le système engendré par la procédure.

Le système REVE 3 [Kirchner-Kirchner-85r] est une réalisation de cette méthode dans les trois cas de E égal à la théorie vide, associatif-commutatif ou commutatif. Quatre types de réécritures sont implantés:

Knuth-Bendix

E = ø

R toutes les règles

→

F_E = {«}ø filtrage

dans la théorie vide

$s =_{EUR} t$

ssi $s \downarrow t$

Huet

E ≠ ø

R: règles linéaires gauche

→

$s =_{EUR} t$

ssi $s \rightarrow_R^* u =_E v \leftarrow_R^*$

Peterson et Stickel

E: axiomes linéaires

R: règles

$\rightarrow_{R,E}$

F_E = $\{«\}_E$ filtrage dans

la théorie E

$s =_{EUR} t$

ssi $s \rightarrow_{R,E}^* u =_E v \leftarrow_{R,E}^*$

Jouannaud

E non-vide quelconque

Rl: règles linéaires gauche

Rn: règles non-linéaires gauche

$\rightarrow_{RlURn,E}$

F_E est {«}ø si la règle

est linéaire et $\{«\}_E$ sinon.

$s =_{EUR} t$

ssi $s \rightarrow_{RlURn,E}^* u =_E v \leftarrow_{RlURn,E}^*$

4. Logique du premier ordre

4. Logique du premier ordre

Dans ce chapitre nous rappelons les notions essentielles de la logique du premier ordre ou calcul des prédicats ou calcul des prédicats du premier ordre (CP1) [Shoenfield-67] et nous introduisons les notations utilisées par la suite.

4.1. Syntaxe de la logique du premier ordre

Dans ce paragraphe nous définissons la syntaxe de la logique du premier ordre.

Définition 4.1 -- Un ensemble de **symboles de relation** $\mathcal{R}$ est un ensemble de symboles disjoint de l'ensemble des symboles de fonction F et de l'ensemble des variables X, on les appelle aussi **symboles de prédicat**. Les symboles de relation sont désignés par les lettres P, Q, R...

○

Comme à tout symbole de fonction, on fait correspondre à tout symbole de relation R un entier appelé aussi **arité** et noté **ar**(R). Si on désire mettre en évidence l'arité d'un symbole de relation on note celui-ci $R_{ar(R)}$

Définition 4.2 -- On appelle **formule atomique** ou **atome** tout terme de la forme:

$$R_i(t_1,\ldots,t_k)$$

avec $k = ar(R)$ et $\forall\, i \in [1,k]\ \ t_i \in M(F,X)$.

L'ensemble des formules atomiques est note $At(\mathcal{R},F,X)$ ou tout simplement **At**.

○

Il est facile de voir que toutes les définitions concernant les termes peuvent s'appliquer aux formules atomiques. Pour tout atome A nous avons:

- L'ensemble des variables $V(A) = \bigcup_{k=1}^{i} V(t_k)$
- Pour toute substitution σ de Subs (l'ensemble de toutes les substitutions de M(F,X)), nous avons:

$$\sigma(A) = R(\sigma(t_1), \ldots, \sigma(t_i)) \in At$$

- Deux atomes A1 et A2 sont unifiables si et seulement si il existe une substitution σ telle que $\sigma(A1) = \sigma(A2)$. Une condition nécessaire pour que deux atomes soient unifiables est qu'ils aient le même symbole de relation.

Nous définissons les **formules** (ou **prédicats**) du premier ordre comme suit.

Définition 4.3 -- Soient A et B deux atomes, par définition:

A est une formule,

$\neg$ A est un formule (se lit non A),

A $\vee$ B est une formule (A ou B),

$\exists$ x A est une formule (il existe x A).

○

Les autres symboles tels que $\forall$ (quelque soit), $\wedge$ (et), $\Rightarrow$ (implique)... ne sont que des abréviations d'écriture, pour ϕ et ψ deux formules, nous avons:

$\phi \Rightarrow \psi$	abréviation de	$\neg\phi \vee \psi$
$\phi \wedge \psi$		$\neg(\phi \Rightarrow \neg\psi)$
$\phi \Leftrightarrow \psi$		$(\phi \Rightarrow \psi) \wedge (\psi \Rightarrow \phi)$
$\forall\, x\ \phi$		$\neg\exists\, x \neg \phi$

○

Notation -- L'ensemble des formules du premier ordre est noté CPl(R,F,X) ou tout simplement **CPl**.

4.2. Sémantique du calcul du premier ordre

Définition 4.4 -- Une **interprétation** I du calcul des prédicats est un triplet (D_I, F_I, R_I) où

- D_I est un ensemble non vide appelé **domaine de l'interprétation**,
- F_I est un ensemble d'applications dans D_I associées aux symboles de fonction de F, avec l'arité correspondante.
- R_I est un ensemble de relations dans D_I associées aux symboles de relation de $\mathcal{R}$, avec l'arité correspondante.

Nous avons:

$f \in F$, $I(f)\colon D_I^{ar(f)} \to D_I$

$R \in \mathbb{R}$, $I(R)\colon D_I^{ar(R)} \to \mathbb{B} = \mathbb{B}$

où $\mathbb{B}$ est égale à {0,1} c'est-à-dire {faux,vrai}

○

L'interprétation permet de donner un sens à une formule A, en remplacant dans A, le $\vee$ par "ou", $\neg$ par "non" etc, on obtient un prédicat qui est vrai sur un certain sous-ensemble A_I de D_n^I si la formule A dépend des variables libres $x_1,..,x_n$. C'est ce que nous précisons ici-dessous.

Définition 4.5 -- Une **valuation** ν d'une interprétation I est une application de l'ensemble des variables X dans le domaine D_I de I

$$\nu\colon X \to D_I,$$

qui peut être étendu à M(F,X) et CPI par morphisme (à cause du caractère libre de M(F,X)). Nous avons:

$$\forall f \in F,\ \nu(f(t_1,\ldots,t_k)) = I(f)(\nu(t_1),\ldots,\nu(t_k))$$
$$\forall R \in \mathbb{R},\ \nu(R(t_1,\ldots,t_k)) = I(R)(\nu(t_1),\ldots,\nu(t_k))$$

ν associe donc à chaque terme t de M(F,X) un élément $(I,\nu)(t)$ de D_I, et à chaque atome A une valeur de vérité $(I,\nu)(A)$ dans $\mathbb{B}$.

○

Définition 4.6 -- La **valeur** d'une formule ϕ dans une interprétation I, notée $val_I(\phi)$ est par définition l'ensemble des valuations ν tel que $(I,\nu)(\phi) = 1$.

La valeur $(I,\nu)(\phi)$ se définit par reccurrence sur la structure de ϕ, on

connait la valeur si ϕ est un atome et on pose

$$(I,\nu)(\phi \vee \phi') = (I,\nu)(\phi) + (I,\nu)(\phi') \quad (+ \text{ booleen})$$

$$(I,\nu)(\neg\phi) = \overline{(I,\nu)(\phi)}$$

$(I,\nu)(\exists x\ \phi) = 1$ si et seulement si pour une valuation ν' différente de ν seulement sur x on a $(I,\nu')=1$.

○

Nous disons qu'une formule ϕ est **universellement quantifiée** si et seulement si toutes ses variables sont dans la portée de $\forall$. (appelée aussi **cloture universelle**). par exemple la formule

$$\neg\forall x\ P(x)$$

n'est pas universellement quantifiée.

Nous définissons la valeur de vérité d'une formule.

Définition 4.7 -- On dit qu'une formule ϕ universellement quantifiée est **vraie** dans une interprétation I et on le note $I \vdash \phi$ si et seulement si pour toute valuation ν $(I,\nu)(\phi) = 1$, sinon la valeur de vérité de ϕ est **faux** ($I \nvdash \phi$). Pour qu'une formule sans variable libre (on dit aussi close ou fermée), soit vraie dans une interprétation I, il suffit d'avoir $(I,\nu_0)(\phi)=1$, où ν_0 est l'unique valuation de domaine $\emptyset$.

○

Définition 4.8 -- On dit qu'une formule ϕ est **valide**, si et seulement si ϕ est vrai dans toute interprétation I, $I \vdash \phi$.

ϕ est **satisfiable** (ou non contradictoire) si et seulement s'il existe une interprétation I telle que $I \vdash \phi$.

ϕ est **insatisfiable** si et seulement si pour toute interprétation I, $I \nvdash \phi$.

Si ϕ est close, ϕ est valide si et seulement si $\neg\phi$ est insatisfiable.

4.3. Rappel sur le calcul des prédicats

Les prédicats peuvent être considérés comme les éléments d'un langage de vocabulaire:

- Variable X = {x, y, z...},
- F ensemble des symboles de fonctions. $F = F_1 \cup F_2 \ldots F_n$, F_i est l'ensemble des symboles de fonction d'arité i.
- L'ensemble des symboles de relation $\mathbb{R}$.
- Les symboles booléens: {1, 0, $\vee$, $\wedge$}
- Les symboles de quantificateurs:
  $Q = \{\forall_x, x \in X\} \cup \{\exists_x, x \in X\}$
- Les ponctuations pour la lisibilité de l'écriture: (,), ,.

Dans le langage des prédicats on distingue plus particulièrement:

Les termes

Les eléments de M(F,X).

Les atomes

$At = \{R(t_1,t_2,..,t_i),\ R \in \mathbb{R},\ t_k \in M(F,X)\}$

Les littéraux

Ce sont les atomes et les négations des atomes.

*Les formules sans quantificateurs et **CPO** (CP-zero)*

Ces formules présentent de grandes analogies avec celles du calcul des propositions, les atomes jouant le rôle de variables propositionnelles. Par exemples

$$P(x,y) \vee \neg P(y,x)$$

est une formule sans quantificateur associée à la formule

$$a \vee \neg b$$

du calcul des propositions.

Les formules

$$\forall x\ P(f(x),\ y)) \vee \exists x\ R(y)$$

où P un symbole relationnel et f un symbole fonctionel, x variable.

Les formules prénexes

Les formules où les symboles $\forall$ et $\exists$ sont en tête

$$\forall x\ \exists y\ (P(g(x),x) \vee R(f(y))$$

Toute formule est équivalente à une formule sous forme prénexe.

Les clauses

Une clause est une formule sans quantificateur qui est une disjonction de littéraux

$$P(f(x),y) \vee \neg R(g(h))$$

Pour une formule A, il existe un ensemble de clauses C(A) tel que A est contradictoire si et seulement si C(A) l'est. Cette ensemble est obtenu par la suite de transformations suivantes qui conservent la satisfiabilité et l'insatisfiabilité.

Skolémisation

La skolémisation est une transformation d'une formule ϕ du premier ordre (en une formule purement universelle appelée **forme de Skolem**), qui conserve la satisfiabilité (et l'insatisfiabilité). En considérant qu'une formule est sous forme prénexe (ce qui est toujours possible), cette transformation consiste à:

* renommer les variables quantifiées plusieurs fois

* supprimer $\exists x$ et remplacer x par $\rho(x_1,..,x_k)$, où $x_1,..x_k$ sont les variables libres de ϕ et les variables universellement quantifiées précédant x dans la liste des quantificateurs et où ρ est un nouveau symbole de fonction appelé fonction de Skolem.

La forme de Skolem d'une formule ω est notée $\hat{\omega}$

La forme de Skolem d'une formule universellement quantifiée, ne contient pas de variables.

Une forme de skolem $\hat{\omega}$ est satisfiable si et seulement si $\forall x_1..\forall x_n\ \omega$ (la formule universellement quantifiée) est satisfiable.

Il ne peut pas exister des méthodes générales permettant de dire si oui ou non une formule du CP1 est vraie. En effet Church (1936) et Turing (1936) ont montré indépendamment que le CP1 est indécidable.

Si une formule est vraie, il existe des approches pour le montrer. Ces approches sont de deux types:

Méthodes positives

montrer la validité de la formule.

Méthodes négatives

montrer que la négation de la formule est insatisfiable.

Les notions présentées jusqu'à présent sont inutilisables en démonstration automatique. En effet, nous faisons appel à des notions sémantiques (liées à la signification) qui sont difficiles, voire impossibles à manipuler par le raisonnement automatique. Il s'agit donc de trouver des moyens syntaxiques (liés à la forme) pour montrer l'insatisfiabilité d'un ensemble de formules A. Une méthode syntaxique consiste à se donner des règles dites **d'inférence**, qui permettent d'engendrer, à partir de A de nouvelles formules, jusqu'à ce que l'une d'elles soit insatisfiable. On dit que la preuve de A se fait par **réfutation**.

L'utilisation des méthodes syntaxiques pose le problème de la légitimité de la méthode: est elle équivalente à la méthode sémantique? Une méthode est dite **correcte** si et seulement si, utilisant cette méthode, lorsqu'elle aboutit à une réfutation de A, A est effectivement insatisfiable. Une méthode est dite **complète** si de plus, lorsque A est insatisfiable alors la méthode permet de réfuter A.

En considérant les formules sous forme clausale, Robinson [Robinson-65], s'appuyant sur les résultats de Herbrand, a montré la complétude de la méthode de **résolution**. Cette méthode ne contient qu'une seule règle d'inférence appelée **résolution**. Un ensemble A de clauses est insatisfiable si on arrive à dériver à partir de A (par résolution) la clause vide (aucun littéral) qui est trivialement insatisfiable.

Cette méthode a permis la mise en oeuvre des premiers programmes effectifs de démonstration automatique. Plusieurs restrictions ont été

développées pour limiter le grand nombre de clauses engendrées qui ne participent pas à la réfutation. Une étude des différentes méthodes peut être trouvée dans [Loveland-78].

Dans le cas particulier des clauses constituées d'un seul littéral et où les littéraux sont des équations (cas des théories équationnelles), la procédure de Knuth et Bendix [Knuth-Bendix-70] est une méthode efficace pour engendrer, à partir d'un ensemble d'équations initial, d'autres équations logiquement déductibles des premières. Plusieurs implantations de la procédure de Knuth-Bendix ont été réalisées, nous donnons dans le chapitre 5 une implantation qui prévilégie la règle de simplification.

Utilisant la procédure de Knuth-Bendix, Hsiang a donné une méthode réfutationnelle complète pour la preuve dans le calcul du premier ordre [Hsiang-82]. Nous décrivons cette méthode et son implantation dans le paragraphe suivant.

4.4. Méthode de Hsiang pour la preuve du CP1

La méthode de Hsiang pour la preuve dans le calcul des prédicats du premier ordre est une méthode se basant sur les techniques de réécriture. Cela lui donne l'avantage de limiter l'espace de recherche. Cette méthode utilise la règle d'inférence de simplification qui n'existe pas en résolution.

4.4.1. Système BA

Hsiang [Hsiang-82] spécifie l'algèbre booléenne en utilisant le "ou exclusif" (+) à la place de l'opérateur "ou" ($\vee$). On note le "et" par $*$, non par $\neg$, vrai par 1, faux par 0. En exprimant l'idempotence de $*$ et la nilpotence de +, et en considérant + et $*$ associatif-commutatif, nous avons le système de réécriture BA suivant:

$$
\begin{aligned}
x \vee y &\rightarrow x*y + x + y \\
x \Rightarrow y &\rightarrow x*y + x + 1 \\
x \Leftrightarrow y &\rightarrow x + y + 1 \\
\neg x &\rightarrow x + 1 \\
x + 0 &\rightarrow x \\
x + x &\rightarrow 0 \\
x*1 &\rightarrow x \\
x*x &\rightarrow x \\
x*0 &\rightarrow 0 \\
x*(y + z) &\rightarrow x*y + x*z
\end{aligned}
$$

Ce système est convergent. La preuve de convergence est laissée à l'implantation préférée du lecteur (il faut un test de convergence pour la réécriture associative-commutative, par exemple REVE 3). Par conséquent, ce système nous donne une procédure de décision pour le problème du mot dans le calcul des propositions (CPO). La forme normale d'une formule (considérée comme un terme) est donc le terme 1, le terme 0 ou un autre terme, qui correspondent respectivement à la validité, l'insatisfiabilité ou la satisfiabilité de la formule considérée.

Exemple 4.1 -- pour prouver la formule $p \Rightarrow p$, il suffit de normaliser ce terme dans le système BA, nous avons la séquence de normalisation suivante obtenue par le système REVE:

$$
\begin{aligned}
& p \Rightarrow p \\
& (p * p) + 1 + p \\
& 1 + p + p \\
& 1 + 0 \\
& 1
\end{aligned}
$$

et pour prouver $p \Rightarrow (\neg(p) \Rightarrow q)$ nous avons la séquence suivante:

$p \Rightarrow (\neg(p) \Rightarrow q)$
$((\neg(p) \Rightarrow q) * p) + 1 + p$
$(((\neg(p) * q) + 1 + \neg(p)) * p) + 1 + p$
$((1 + \neg(p)) * p) + (\neg(p) * p * q) + 1 + p$
$(1 * p) + (\neg(p) * p) + (\neg(p) * p * q) + 1 + p$
$(\neg(p) * p) + (\neg(p) * p * q) + 1 + p + p$
$(\neg(p) * p) + (\neg(p) * p * q) + 0 + 1$
$(\neg(p) * p) + (\neg(p) * p * q) + 1$
$((1 + p) * p) + (\neg(p) * p * q) + 1$
$(1 * p) + (\neg(p) * p * q) + (p * p) + 1$
$(\neg(p) * p * q) + (p * p) + 1 + p$
$((1 + p) * p * q) + (p * p) + 1 + p$
$(((1 * p) + (p * p)) * q) + (p * p) + 1 + p$
$(1 * p * q) + (p * p) + (p * p * q) + 1 + p$
$(p * p) + (p * p * q) + (p * q) + 1 + p$
$(p * p * q) + (p * q) + 1 + p + p$
$(p * p * q) + (p * q) + 0 + 1$
$(p * p * q) + (p * q) + 1$
$(p * q) + (p * q) + 1$
$0 + 1$
1

on peut aussi montrer que $p \vee (p*q) = p$ en normalisant le membre gauche

$p \vee (p * q)$
$(p * p * q) + (p * q) + p$
$(p * q) + (p * q) + p$
$0 + p$
p

4.4.2. Preuve par réfutation

Bien que cette méthode soit complète pour le calcul des propositions, elle présente un certain nombre d'inconvenients:

(1) Dans le cas des formules contenant un grand nombre de symbole $\Rightarrow$, les termes booléens peuvent sont, en général, très grande taile.

(2) La normalisation à l'aide de BA est insuffisante pour traiter le cas général des formules du premier ordre, à cause notamment de la présence de variables.

Ces problèmes sont contournés en utilisant une preuve par réfutation:

(1) Pour prouver la validité d'une formule $\phi = b_1 \vee .. \vee b_n$ du calcul des prédicats du premier ordre, on skolémise sa négation, et on la met sous forme clausale $\neg c_1 \wedge \neg c_2 \wedge .. \wedge \neg c_n$.

(2) Transformation des clauses en règles de réécriture $c_i \rightarrow 0$. le système BA est utilisé pour calculer la forme normale des c_i.

(3) Afin de limiter la taille des termes, les règles de la forme $p_1 * ... * p_k + 1 \rightarrow 0$ sont transformées en l'ensemble des règles suivantes: $p_1 \rightarrow 1, .., p_k \rightarrow 1$. Cette transformation conserve la satisfiabilité. D'autres méthodes de tranformations de clauses en règles de réécriture peuvent être trouvées dans [Hsiang-Josephson-83]

(4) La situation particulière de la logique du premier ordre permet d'utiliser un algorithme appelé BN-unification [Hsiang-82], plus adapté et plus efficace que l'algorithme d'unification AC le plus général. Il tient compte du fait que les symboles + et * n'ont pas pour sous-termes immédiats une variable, et du fait qu'un atome apparaît dans un même monôme au plus une fois, à cause de la propriété d'idempotence de *.

Nous donnons maintenant les règles d'inférence de la méthode de preuve par réfutation appelée N-strategie.

Les règles booléennes sont divisées en trois ensembles, N l'ensemble des règles de la forme $p_1*..*p_n \rightarrow 0$, P celles de la forme $p_1*..*p_n \rightarrow 1$, et 0 (pour "others") l'ensemble des autres règles. Pour δ égal à 1 ou 0, une règle de la forme $p+1 \rightarrow \delta$, est considérée comme la règle $p \rightarrow \delta+1$ ($\delta+1$ est égal à 0 si $\delta=1$ et 1 sinon). S est l'ensemble des règles simplificatrices.

La notation N/P, représente une BN-unification-superposition (Def 3.6) d'une règle de N sur une règle de P.

N_1) (Réduction d une règle de N, P, 0 ou S)

$$\frac{N \cup \{g \rightarrow \delta\}, P, 0, S}{N, P, 0, S \cup \{g' \rightarrow \delta\}} \quad \bullet\ g \rightarrow g' \text{ (par S)}$$

N_2) (Fin de Réduction 1)

$$\frac{N, P, 0, S \cup \{s \rightarrow 0\}}{N \cup \{s \rightarrow 0\}, P, 0, S} \quad \bullet \text{ plus rien n'est simplifiable par } s \rightarrow 0.$$

N3) (Fin de Réduction 2)

$$\frac{N, P, 0, S \cup \{s \rightarrow 1\}}{N, P \cup \{s \rightarrow 1\}, 0, S} \quad \bullet \text{ plus rien n'est simplifiable par } s \rightarrow 1.$$

N4) (Superposition 1)

$$\frac{N, P, 0, \emptyset}{N, P, 0, S \cup \{s \rightarrow \delta\}} \quad \bullet\ s \rightarrow \delta \in N/P \text{ ou } s \rightarrow \delta \in N/0 \text{ et s est un N-terme ou un P-terme}$$

N5) (Superposition 2)

$$\frac{N, P, 0, \emptyset}{N, P, 0 \cup \{s \rightarrow \delta\}, \emptyset} \quad \bullet\ s \rightarrow \delta \in N/0 \text{ et s est un 0-terme}$$

N6) (Contradiction)

$$\frac{N, P, 0, S \cup \{1 \rightarrow 0\}}{\text{CONTRADICTION}}$$

Lorsque ces règles d'inférence engendrent "CONTRADICTION", l'ensemble des clauses considéré au départ est insatisfiable, dans les autres cas il est satisfiable. Nous ne donnons pas de preuve ici, elles peuvent être trouvées dans la thèse de Hsiang [Hsiang-82] (voir aussi celle de J. Durand [Durand-82]).

Exemple 4.2 -- Reprenons l'exemple précédent, $p \Rightarrow p$, l'ensemble des clauses obtenues par négation de la formule est $\{p \rightarrow 1, p+1 \rightarrow 0\}$ qui est transformé en $\{p \rightarrow 1, p \rightarrow 0\}$; par superposition, la contradiction est immédiatement trouvée.

∇

La N-stratégie peut être généralisée au cas où les prédicats sont définis sur un domaine particulier, spécifié par un système convergent de règles de réécriture. Ce système est pris en compte avec les règles booléennes. Les règles d'inférence suivantes sont alors ajoutées:

Soit T le système de règles de réécriture specifique du domaine.

RN_1) (Superposition 3)

$$\frac{T, N, P, 0, S}{T, N, P, 0, S \cup \{s \rightarrow \delta\}} \quad \bullet\ s \rightarrow \delta \in T/N \text{ ou } T/P \text{ ou } N/0 \text{ et s est un N-terme ou un P-terme}$$

RN_2) (Superposition 4)

$$\frac{T,N,P,O,\emptyset}{T,N,P,O\cup\{s\rightarrow\delta\},\emptyset} \quad * \; s\rightarrow\delta \in T/O \text{ et } s \text{ est un O-terme}$$

Nous avons réalisé une implantation de cette méthode comme extension du système REVE 3. Nous décrivons maintenant quelques aspects de cette implantation.

* La simplification (règle N_1) est privilégiée par rapport aux autres: tant que l'ensemble S n'est pas vide, on applique la règle N_1.

* Les ensembles de règles sont toujours triés par ordre croissant de taille des termes.

* Utilisation de la factorisation des termes au cours du calcul des paires critiques: par exemple pour superposer la règle r_1: $M(x,i(y),z)P(x)P(y)P(z)+M(x,i(y),z)P(x)P(y)\rightarrow 0$ avec la règle r_2: $P(i(b))\rightarrow 0$, il est inutile d'unifier P(i(b)) avec P(x) ou P(y) car r_1 peut se factoriser en $P(x)P(y)(M(x,i(y),z)P(z)+M(x,i(y),z))$, et toute superposition avec un sous-terme de P(x)P(y) donne une paire critique triviale. Dans ce cas la seule superposition possible est celle de P(i(b)) avec P(z) qui donne la N-paire critique $\langle M(x,i(y),i(b))P(x)P(y),0\rangle$ [Durand-84]

* Nous avons utilisé aussi les possibilités du langages CLU [Liskov,etal.-81] (langage d'implantation de la méthode). Cela permet pour deux termes booléens b1 et b2, d'utiliser l'écriture "b1 + b2", (rep b1 * b2) pour le résultat de l'opération booléenne + (rep *) avec

utilisation du système BA pour la normalisation. L'ensemble des règles BA est implanté par des procédures dans le système et utilisé implicitement dans toutes les opérations. Il n'est pas considéré comme système de réécriture pour des raisons évidentes d'efficacité.

L'utilisation de cette implantation, en plus de la méthode de preuve qu'elle offre dans le CP1, est envisagée à l'intérieur du système REVEUR4 [Remy-Zhang-84], en particulier pour tester l'équivalence des contextes des règles conditionnelles, qui sont des formules du premier ordre.

5. Algorithme de Complétion

5. Algorithme de Complétion

5.1. Introduction

L'efficacité de la procédure de complétion est étroitement liée au mécanisme de simplification des règles et des équations, ce qui limite le nombre des paires critiques engendrées.

Récemment, plusieurs auteurs ont donné des méthodes pour améliorer l'efficacité de la procédure de complétion [Winkler-Buchberger-83,Kuechlin-85,Kapur,etal.-85]. Ces méthodes sont toutes basées sur le principe de la limitation du calcul des paires critiques. Toutes ces approches ont été unifiées par le formalisme des "critères de paires critiques" donné par Bachmair et Dershowitz [Bachmair-Dershowitz-86b].

Nous donnons dans le paragraphe suivant une procédure de complétion appelée SKB, où la notion de simplification est privilégiée par rapport celle de génération de paires critiques. Nous pensons en effet que c'est la simplification qui apporte toute sa puissance aux systèmes de réécriture et que c'est l'une des clées de l'efficacité d'une procédure de complétion:

- Lorsqu'on génère une paire critique, on augmente le système d'une équation; lorsqu'on simplifie, on remplace le système par un système plus simple.

- La génération de paires critiques utilise un algorithme d'unification, la simplification utilise un algorithme plus rapide de filtrage.

L'algorithme de complétion SKB est un algorithme original différent de celui de Huet [Huet-81]. Nous donnons sa preuve de correction totale en utilisant l'outil d'ordre sur les preuves équationnelles de Bachmair-Dershowitz-Hsiang [Bachmair,etal.-86]. Nous décrivons son implantation dans le système SBREVE, réalisé à la State University of New York en collaboration avec J. Hsiang. Sbreve est une extension de la version 2.4 du système REVE [Forgaard-84]. Nous montrons sur quelques exemples comment l'implantation de l'algorithme SKB est plus efficace que d'autres implantations d'algorithme de complétion.

5.2. Algorithme de complétion rapide (SKB)

Nous décrivons dans cette partie notre algorithme de complétion, où la simplification joue un rôle privilégié. Notre premier souci est l'efficacité, mais cela n'est pas au détriment de la simplicité de l'algorithme. La règle de simplification a été toujours considérée comme nécessaire pour la convergence du processus, elle est aussi vitale pour son efficacité.

Dans la procédure SKB, comme dans toutes les procédures de complétion, pour prouver la terminaison de la réécriture nous avons besoin d'un ordre de réduction sur les termes, que l'on note <. Nous ne nous étendons pas dans cette thèse sur les méthodes et les problèmes de terminaisons de systèmes de réécriture de termes, plusieurs recherches se font dans ce

domaine [Bachmair-Dershowitz-86a,BenCherifa-86,Gnaedig-86].. nous nous limitons à donner la définition d'un ordre de réduction nécessaire pour l'orientation des termes dans la procédure de complétion.

Définition 5.1 -- **Un ordre de réduction** < est un ordre partiel sur l'ensemble des termes tel que:

1) > est bien fondé (il n'existe pas de séquence infinie)

2) > est clos par substitution (on dit aussi stable par instantiation) $(M>N \Rightarrow \sigma M>\sigma N)$

3) > a la propriété de F-compatibilité (on dit aussi monotonicité [Dershowitz-85] ou "est clos par remplacement de sous-termes" [Hullot-80] $(M>N \Rightarrow f(..M..)>f(..N..))$ ou "précongruence" en mathématiques.

Une sous classe d'ordre de réduction est la classe des ordres de simplification (la propriété de bon fondation est remplacée par celle de "sous-terme" f(..M..)>M). Plusieurs ordre de simplification: RPO [Dershowitz-82], RDO, [Jouannaud,etal.-82].. sont implantés dans le système REVE et permettent la preuve automatique de terminaison pour une grande classe d'exemples.

Signalons que la terminaison est une propriété indécidable en général (i.e. il n'existe pas d'algorithme qui permette de dire si oui ou non un système termine).

5.2.1. Règles d'inférence

L'algorithme de complétion SKB est basé sur des règles d'inférence simples pour la transformation du triplet (R,E,S) où R est l'ensemble des règles déjà superposées sur lui-même, E l'ensemble des règles ou équations déjà utilisées comme simplificateurs et S l'ensemble des règles non encore utilisées comme simplificateurs. Nous ne partageons pas les ensembles suivant les équations et les règles comme dans [Bachmair-Dershowitz-86b], car pour nous, une équation n'est autre qu'une règle non encore orientée, ou au pire non orientable, (dans ce cas on espère qu'à une étape ultérieure de l'algorithme, on aura une règle qui la simplifie et donc qu'elle disparaitra du système). E peut donc contenir des règles ou des équations, R ne contient que des règles. Si après un certain nombre d'itérations, E devient vide, R est alors un ensemble convergent de règles.

Les règles d'inférence utilisées dans l'algorithme de complétion sont basées sur les mécanismes suivants:

S1▶ La réduction par les simplificateurs de S, des règles de R, E et S. Ce mécanisme peut engendrer de nouveaux simplificateurs à partir des règles ou équations simplifiées.

S2▶ Une règle de S qui ne simplifie plus est transférée dans E.

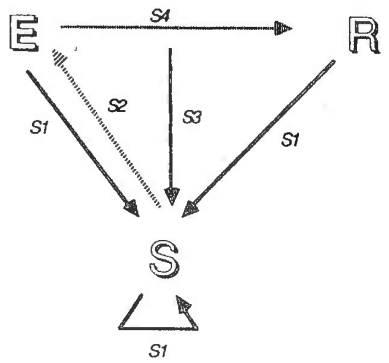
S3▶ Le calcul des paires critiques des règles de E avec celles de R.

S3▶ L'orientation de nouvelles paires critiques et leur utilisation comme simplificateurs.

S4▶ Le transfert dans R d'une règle de E dont les paires critiques ont été calculées.

Les règles d'inférence précises utilisées sont données au paragraphe suivant.

Les transferts possibles entre R,E et S peuvent être schématisés par le graphe suivant:



5.2.2. Algorithme SKB et preuve de correction

Nous présentons la complétion SKB en utilisant le formalisme de [Bachmair,etal.-86], et nous prouvons sa correction totale par la méthode des ordres sur les preuves équationnelles de Bachmair-Dershowitz-Hsiang [Bachmair,etal.-86].

Nous définissons t>>t' par: **t>>t'** si et seulement si

- $t_{|u} = \sigma t'$ et
- $t \neq t'$ à un renomage près (c'est-à-dire $u \neq \varepsilon$ ou $\sigma \neq Id$)

La théorie des preuves SKB est définie par les règles d'inférence suivantes où:

- E et S sont des ensembles de règles (orientées ou non),
- R est un ensemble de règles orientées,

T désigne indifféremment S, E ou R

- et > un ordre de réduction,

S1.1) Simplification du membre droit d une règle utilsant S

$$\frac{TU\{s\to t\},S}{TU\{s\to t'\},S}$$

- $t \to_S t'$

S1.2) Simplification du membre gauche d une règle utilsant S

$$\frac{TU\{s\to t\},S}{T,SU\{s'=t\}}$$

- $s \to_{l\to r} s'$ avec $l\to r \in S$ et $s >> t$

S1.3) Simplification par R

$$\frac{E,R,SU\{s=t\}}{E,R,SU\{s'=t'\}}$$

- $s \to_R s'$ $t \to_R t'$

S2) Fin de Réduction

$$\frac{E,R,SU\{s\to t\}}{EU\{s\to t\},R,S}$$

- plus rien n'est simplifiable par $s\to t$.

S3) Superposition

$$\frac{E,R,\emptyset}{E,R,\{s=t\}}$$

- $u \to_R s$ et $u \to_R t$ où u est une superposition d'une règle de R et une règle de E.

S4) Orientation

$$\frac{E,R,SU\{s=t\}}{E,R,SU\{s\to t\}}$$

- $s > t$

S5) Fin de superposition

$$\frac{EU\{l\to r\},R,\emptyset}{E,RU\{l\to r\},\emptyset}$$

S6) Elimination des paires critiques triviales

$$\frac{E,R,SU\{s=s\}}{E,R,S}$$

Nous notons par $(E,R,S) \Rightarrow_{SKB} (E',R',S')$ l'application d'une ou plusieurs règles d'inférence (Si). $\Rightarrow_{SKB}$ est appelé **SKB-dérivation.**

Une **preuve** de s=t dans (E,R,S) est une suite de termes

$$s_0 - \ldots\ldots - s_n$$

où s_0=s et s_n=t et

- est soit $\longleftrightarrow_S$ $\to_R$ $\to_E$ $\leftarrow_R$ $\leftarrow_E$

Nous allons montrer d'abord la correction des règles d'inférences

ensuite la complètude.

Le lemme suivant montre que si s=t est prouvable dans (E,R,S), alors s=t est prouvable dans (E',R',S') avec (E,R,S) $\Rightarrow_{SKB}$ (E',R',S'), ce qui prouve la correction des règles d'inférence SKB.

Lemme 5.1 -- (Lemme de correction)-- Si (E,R,S) $\Rightarrow_{SKB}$ (E'R'S') alors les relations de congruence $\longleftrightarrow_{EURUS}$ et $\longleftrightarrow_{E'UR'US'}$ sont les mêmes.

Nous montrons ensuite que la construction d'un système convergent R à partir d'un ensemble d'équations E est possible, sous certaines conditions, en utilisant les règles d'inférence de la théorie des preuves SKB.

$$(\emptyset, \emptyset, S) \ldots \Rightarrow_{SKB} \ldots (\emptyset, R, \emptyset)$$

En d'autres termes nous montrons que toutes les preuves de la forme

$$s_1 \longleftrightarrow_{EURUS} \cdots \longleftrightarrow_{EURUS} s_n$$

peuvent être transformées en des preuves de la forme

$$s_1 \rightarrow_R^* u \leftarrow_R^* s_n$$

que l'on applelle preuve par réécriture.
Ce résultat est dû au fait que les preuves de s=t dans (E',R',S') sont "plus simples" que celle dans (E,R,S). Nous formalisons cette notion en définissant un ordre sur ces preuves.

Définissons dans ce qui suit l'ordre sur les preuves équationnelles [Bachmair,etal.-86].

Définition 5.2 -- Un ordre $>_{TP}$ est un ordre preuves si et seulement si

- $>_{TP}$ est bien fondé (pas de chaînes infinie)
- Stable: si P=(s,..s_i,..t) et P'=(s,..s'_i,..t) sont deux preuves telles que $P >_{TP} P'$, alors pour tout terme u et toute substitution σ

$$(u[\sigma s],..u[\sigma s_i],..u[\sigma t]) >_{TP} (u[\sigma s],..u[\sigma s'_i],..u[\sigma t]),$$

- compatible avec la théorie des preuves TP: si (E,R,S) $\Rightarrow_{TP}$ (E',R',S') alors pour toute preuve P de $s \longleftrightarrow_{EURUS} t$, il existe une preuve P' de $s \longleftrightarrow_{E'UR'US'} t$, telle que $P \geq_{TP} P'$.

O

Nous allons construire un ordre de preuves à la Bachmair-Dershowitz-Hsiang, basé sur un ordre de réduction >. Nous définissons d'abord une mesure de compléxité c_i sur une étape élémentaire $(s_i - s_{i+1})$ d'une preuve P de s=t, et nous définissons un ordre $>_c$ sur ces compléxités, qui bien sûr est construit grâce à l'ordre de réduction >.

Considérant ensuite une preuve comme un multiensemble d'étapes élémentaires de preuves, on associe à chaque preuve P le multiensemble $M(P)=\{c_1,..,c_n$ des compléxités des étapes élémentaires de preuves de P. D'où l'ordre sur les preuves:

$$P >_{TP} P' \text{ si et seulement si } M(P) >_m M(P')$$

où $>_m$ est l'extension multiensemble [Dershowitz-Manna-78] de $>_c$ qui est connue pour être un ordre bien fondé si $>_c$ est un ordre bien fondé.

Pour notre cas, la théorie des preuves SKB, considérons la mesure de

complexité c suivante:

si $u \leftrightarrow_S v$ alors $c(u,v) = (\{u,v\},\_,\_,\_,S)$

si $u \rightarrow_S v$ par $l \rightarrow r$ à l'occurrence δ alors $c(u,v) = (\{u\},u_{|\delta},l,v,S)$

si $u \rightarrow_R v$ par $l \rightarrow r$ à l'occurrence δ alors $c(u,v) = (\{u\},u_{|\delta},l,v,R)$

si $u \rightarrow_E v$ par $l \rightarrow r$ à l'occurrence δ alors $c(u,v) = (\{u\},u_{|\delta},l,v,E)$

L'ordre $>_c$ est la combinaison lexicographique de:

l'extension multiensemble de l'odre de réduction >,

l'ordre des sous-termes stricts,

l'ordre de filtrage ($>_\sigma$),

l'ordre de réduction >,

et l'ordre sur l'ensemble {E,R,S} définie par S>R>E.

Lemme 5.2 -- L'ordre $>_{SKB}$ associé à $>_c$ est un ordre sur les preuves équationnelles.

Preuve :

L'ordre $>_{SKB}$ est bien fondé puisque > l'est, de même pour la stabilité. Pour prouver la compatibilité de $>_{SKB}$ avec la théorie SKB, nous avons:

- $E,R,SU\{s=t\} \Rightarrow_{SKB} E,R,SU\{s \rightarrow t\}$ par S4

  alors $(s \leftrightarrow t) >_{SKB} (s \rightarrow t)$

  puisque $\{s,t\} >_c \{s\}$

- $E,R,SU\{s=s\} \Rightarrow_{SKB} E,R,S$ par S6

  alors $(s \leftrightarrow s) >_{SKB} (s)$

  puisque $\{s,s\} >_c \emptyset$

- $E,R,\emptyset \Rightarrow_{SKB} E,R,\{s=t\}$ par S3

  alors $(s \leftarrow u \rightarrow t) >_{SKB} (s \leftrightarrow t)$

puisque $\{u\} >_c \{s,t\}$

- si $TU\{s \rightarrow t\},S \Rightarrow_{SKB} TU\{s \rightarrow t'\},S$ par S1.1 avec $t \rightarrow_S t'$

  alors $(s \rightarrow t) >_{SKB} (s \rightarrow t' \leftarrow t)$;

- si $TU\{s \rightarrow t\},S \Rightarrow_{SKB} T,SU\{s'=t\}$ par S1.2 avec $s \rightarrow_{l \rightarrow r} s'$ avec $l \rightarrow r \in S$ et $s >> t$

  alors $(s \rightarrow t) >_{SKB} (s \rightarrow s' \leftrightarrow t)$;

- si $E,R,SU\{s=t\} \Rightarrow_{SKB} E,R,SU\{s'=t'\}$ par S1.3 avec $s \rightarrow_R s'$ et $t \rightarrow_R t'$

  alors $(s \leftrightarrow t) >_{SKB} (s \rightarrow s' \leftrightarrow t' \leftarrow t)$

- si $E,R,SU\{s \rightarrow t\} \Rightarrow_{SKB} EU\{s \rightarrow t\},R,S$ par S2

  alors $(s \rightarrow t) >_{SKB} (s \rightarrow t)$ puisque S>E

- $EU\{l \rightarrow r\},R,\emptyset \Rightarrow_{SKB} E,RU\{l \rightarrow r\},\emptyset$ par S5

  alors $(l \rightarrow r) >_{SKB} (l \rightarrow r)$ puisque E > R

la propriété de compatibilité montre que lorsqu'une règle d'inférence de SKB est appliquée, certaines partie de preuves peuvent être remplacée par des preuves plus simples

□

Nous donnons la condition d'équité (fairness) pour une SKB-dérivation.

Définition 5.3 -- La condition d'équité pour une SKB-dérivation est:

a) $\cap_{j>i} E_i = \emptyset$ pour tout i,

b) si $c=d \cap \in_{j \geq i} CP_j$, alors $s=t \in E_k$, CP_i est l'ensemble de toutes les paires critiques entre les règles de R.

○

Une procédure de complétion SKB est une procédure qui prend en entrée un ordre de réduction >, et un ensemble d'équation S, et utilise les règles d'inférence SKB. Supposons que les SKB-dérivations vérifient la condition

d'équité, nous avons le théorème suivant:

Théorème 5.1 -- Soit SKB une procédure de complétion. Si $s \longleftrightarrow_E t$ et SKB ne s'arrete pas en échec pour la donné d'un ensemble d'équation S et un ordre >, alors SKB engendre (E_i, R_i, S_i) tel que $s \downarrow_{R_i} t$.

5.3. Implantation de SKB

Dans ce paragraphe, nous ajoutons aux règles d'inférence du paragraphe précédent des règles de contrôle pour obtenir un algorithme que nous appelons **SKB**. Signalons que lorsqu'on parle d'orientation ou de non-orientation, il s'agit d'un ordre de réduction choisi une fois pour toute dans la procédure. Si l'on change l'ordre au cours du processus, celui-ci doit être compatible avec l'orientation des règles présentes dans le système parce que la propriété de terminaison est une propriété globale des systèmes de réécriture.

Dans la description des algorithmes, nous utilisons la syntaxe du langage de programmation CLU [Liskov,etal.-81] utilisé pour l'implantation de SKB. L'instruction **continue** reprend la boucle où elle se trouve, et **exit** sort de la boucle.


```
% l'initialisation: R = ø, E = ø, S = l'ensemble des équations
% de l'utilisateur.

SKB = proc(R,E,S: ensemble de règles)
   SIMP(S,E,R)
   % à la sortie de SIMP, S est vide
   répéter jusqu'à E = ø

      • Prendre la première règle l→r de E (E:=E-{l→r})
      • Engendrer UNE paire critique non triviale R-normalisée g=d
        de l→r sur RU{l→r}
        • Si g=d n'est pas orientable alors la mettre dans E
                                              % engendrer une nouvelle
                                              % paire critique
                                            continue
        • SIMP({g→d}US,E,R)
        • Si l→r est simplifiée alors exit  la boucle "Engendrer"
          critiques
        • Si il n'y a plus de paires critiques pour l→r alors R:=RU{l→r}
fin SKB



SIMP = proc(S,E,R: ensemble de règles)
   Tant que S ≠ ø faire
      • Prendre la première équation l=r de S
      • si l=r n'est pas orientable alors E:=EU{l=r}
                                            continue
      • SIMPLF({l→r},S)
      • SIMPLF({l→r},E)
      • SIMPLF({l→r},R)
      • E:=EU{l→r}
fin SIMP



SIMPLF = proc({l→r},T: ensemble de règles)
   répéter jusqu'à ce que toutes les règles de T soient considérées.
      • Prendre une règle g→d de T
      • Si g→d est simplifiée par l→r en g'=d' alors
        S:=SU{g'=d'}
        T:=T-{g→d}
fin SIMPLF
```

Notons que dans cet algorithme les règles sont toujours en R-forme normale, (les règles de R sont donc toujours inter-réduites) car une règle passe toujours par S l'ensemble des simplificateurs avant d'aller dans E et

ensuite dans R. Il est aussi à noter que dans SIMPLF, lorsqu'il n'y a que le membre droit d d'une règle g→d qui est simplifié en d', cette règle n'est pas un nouveau simplificateur et il est donc inutile de la transférer dans S. Par contre, si son membre gauche se réduit en g', g'=d est alors à orienter de nouveau, et à mettre dans S, puisqu'elle devient un simplificateur.

La stratégie du calcul des paires critiques est la suivante: on ne calcule jamais l'ensemble des paires critiques relatives à deux règles en une seule fois, on les calcule une par une et on les utilise immédiatement pour la simplification (elles sont mises dans S). on n'engendre une nouvelle paire qu'après avoir utilisé les précédentes pour simplifier (récursivement) toutes les règles et équations du systèmes. Si l'une des règles engendrant cette paire critique est simplifiée, on arrête le génération des paires critiques.

Il est aussi très important d'utiliser les paires critiques par ordre de taille croissante, les petites d'abord, cela donne plus de chance de simplifier, avant de continuer l'algorithme. Cette remarque est vitale dans l'une des méthodes de preuve issues de l'algorithme de complétion (étudiée plus loin), où le but est de trouver la règle l→0, le plus tôt possible.

Il est possible d'utiliser toutes les règles orientées du système pour normaliser les nouvelles paires critiques, c'est-à-dire, qu'en plus des règles de R, on utilise les règles orientées de E. Si cette stratégie n'est pas utilisée, et s'il existe une règle l→r dans E qui simplifie la paire critique g=d, dans une étape ultérieure de l'algorithme (lorsque l→r est mise dans R), g=d sera simplifiée. Cette stratégie simplifie donc le

système et nous permet de simplifier tout de suite, ce qui peut être simplifiable plus tard dans l'algorithme. Dans notre implantation SBREVE, nous avons utilisé la technique des tables pour implanter les ensembles des règles[Forgaard-84]: les règles sont considérées dans une table H h-codée par le symbole de tête du membre gauche l d'une règle l→r. Cela permet de ne sélectionner que les règles utiles à la réduction d'un terme t de symbole de tête f, à l'occurrence ε, c'est-à-dire de symbole de tête f.

5.4. Experimentation

Les temps donnés dans le tableau suivant sont en secondes. Ce sont des temps CPU sur VAX 785. L'implantation de SKB est réalisée en CLU dans le système SBREVE. Nous l'avons réalisé sur les machines SUN 3/75, et VAX 785. SBREVE contient toutes les possibilitée offertes par REVE. KB est l'implantation de l'algorithme de Knuth-Bendix dans le système REVE [Forgaard-Guttag-84]

Les exemples considérés sont classiques, les spécifications complètes correspondant à ces exemples sont données dans l'appendice 1.

Nous pouvons remarquer qu'en général l'algorithme SKB est plus rapide de 1/3 que KB sur des "petits" exemples (group, fib..). La différence devient plus significative lorsque nous considérons des exemples de plus grande taille (z). Cela montre l'importance de la simplification pour de tels exemples.

	KB	SKB
group	15	9
fib	7	5
ackermann	1	0.6
stack	3	2
z	94	42
pgl	17	1

6. Extension UKB

6.1. Introduction

La procédure de complétion de Knuth-Bendix en tant que procédure de complétion de systèmes de réécriture est une procédure de semi-décision dans le sens où, considérant en entrée un ensemble d'équations E et un ordre sur les termes, elle fournit, s'il existe, un ensemble convergent de règles de réécriture. Elle peut aussi s'arrêter en échec, par exemple s'il n'est pas possible d'orienter une paire critique, ou bien ne pas s'arrêter en ajoutant indéfiniment de nouvelles règles.

Dans plusieurs cas d'échec, on peut encore utiliser la procédure de complétion de Knuth-Bendix pour décider de l'égalité des termes. Citons quelques uns de ces cas d'échec et leurs solutions:

- Il peut arriver que l'ordre utilisé par la procédure de complétion ne soit pas capable de comparer les termes des paires critiques afin d'assurer la terminaison de la réécriture. On peut dans ce cas définir un ordre plus puissant, qui peut orienter ce type de paires critiques [Jouannaud-Lescanne-82,BenCherifa-Lescanne-86,Bachmair-Dershowitz-86a]... Rappelons que la propriété de terminaison est indécidable [Huet-Lankford-78,Dershowitz-85].

- Pour certaines équations comme l'axiome de commutativité (x.y=y.x), il n'est pas possible d'orienter celles-ci sans perdre la propriété de terminaison de la réécriture. Dans ces cas il est possible d'isoler les équations "à problème," et de réécrire modulo ces équations, en utilisant un algorithme d'unification et un algorithme de filtrage modulo ces équations. C'est la méthode générale de réécriture équationnelle présentée dans le chapitre précédent.

- Dans le cas où toutes les paires critiques peuvent être orientées, la procédure de complétion peut ne pas terminer (on dit qu'elle diverge). Dans ce cas on ne sait à aucun moment si elle n'a pas encore trouvé de système convergent ou bien s'il n'existe pas de tel système qui soit fini. Cette divergence de l'algorithme n'est toute fois pas si catastrophique que l'on pourrait le croire. Huet [Huet-81] a montré que la procédure de Knuth-Bendix même dans ce cas de divergence reste une procédure de semi-décision pour l'égalité de deux termes t et t'. Si $t =_E t'$ alors il existe une itération de l'algorithme pour laquelle t et t' auront même forme normale. il suffit donc de calculer à chaque étape les formes normales de t et t' et de vérifier leur égalité. Cette propriété est à la base de certaines méthodes de démonstration automatique en logique du premier ordre [Hsiang-Dershowitz-83,Paul-84].

Dans ce chapitre nous présentons une méthode due à Hsiang et Rusinowitch [Hsiang-Rusinowitch-85a] et implantée dans le système SBREVE, qui est une extension de la procédure de complétion, et donne une procédure de semi-décision pour le problème du mot dans le cas où il n'est pas possible d'orienter une équation au cours de l'algorithme de Knuth-Bendix. Ce cas d'échec pour la procédure de complétion SKB (et pour la procédure de Knuth-Bendix en général) ne l'est plus dans le cas de cette méthode. Cette extension appelée **UKB** (de l'anglais Unfailing Knuth-Bendix) enlève la condition de n'avoir que des règles orientables dans la procédure de complétion de Knuth-Bendix (ou de SKB). L'idée est de garder l'équation g=d non orientable telle quelle et de l'utiliser pour réduire les termes clos, pour lesquels l'ordre sera supposé total: cette réduction est une généralisation de la réduction classique, dans le sens où lorsque g=d est orientable en $g \rightarrow d$, la réduction généralisée est exactement la réduction classique. Cette réécriture nécessitera une notion de "paire critique généralisée," qui étend la notion classique de paire critique.

La procédure de complétion UKB est très similaire à la procédure de complétion de Knuth-Bendix (ou SKB): elle engendre des paires critiques généralisées, les oriente si possible et les utilise pour simplifier les autres règles. Lorsqu'elle s'arrête, avec un ensemble de règles toutes orientables, nous avons un système convergent. Si elle s'arrête avec un ensemble de règles orientées et d'équations, nous avons un système convergent sur les termes clos (tous les termes clos ont une unique forme normale). Un tel système est appelé un **système convergent pour les termes clos.**

UKB peu être adopté au raisonnement par réfutation. Pour montrer que $E \vdash s=t$ où s et t sont des termes non clos (que l'on ne peut décider égaux par la réécriture si seule la convergence close est obtenue), on se ramène à montrer que $\{E, \hat{s} \neq t\}$ est contradictoire, où $\hat{s}$ est la skolémisation de s. Pour cela on cherche à engendrer suffisamment de conséquences

équationnelles de E (paires critiques) pour pouvoir réduire $\hat{s} \neq \hat{t}$ à l'inégalité contradictoire ~~a≠a~~. Si on part avec un ordre total sur les termes clos, les équations non orientables ne posent plus de problème, car le but est de réduire $\hat{s} \neq \hat{t}$ qui est sans variables. Ce qui nous intéresse ce sont donc les instances closes des règles, or celles-ci sont toujours orientables d'après l'hypothèse faites sur l'ordre. Nous revenons en détail sur cette méthode dans le paragraphe 6.3.

6.2. UKB Algorithme de Complétion sans échec

6.2.1. Ordre de simplification total sur les termes clos

Dans UKB, nous avons besoin d'un ordre de simplification total sur les termes clos.

Définition 6.1 -- Un ordre de simplification **fort** est un ordre $>$ sur les termes vérifiant les propriétés suivantes:

1) propriété de sous-terme: $f(..t..) > t$

2) monotonicité: $s>t \Rightarrow f(..s..)>f(..t..)$

3) stabilité par substitution: $s>t \Rightarrow \sigma s>\sigma t$

4) $>$ est total sur l'ensemble des termes clos (sans variables)

O

De manière évidente un ordre de simplification fort est un ordre bien fondé sur les termes (Dershowitz).

Nous remarquons que la plupart des ordres de simplification (RPO, RDO...) peuvent être aisément transformés en des ordres de simplification forts.

Dans tout ce paragraphe $<$ désigne un ordre de simplification fort.

6.2.2. Notions de base

De la même manière que l'algorithme de complétion de Knuth-Bendix, la procédure UKB utilise une notion de paire critique mais celle-ci n'est pas limitée à la superposition des règles. Avec cette nouvelle notion de paire critique appelée paire critique généralisée, il est possible de superposer règles et équations:

Définition 6.2 -- Soient $g[t]=d$ et $l=r$ deux équations, où t est un sous-terme non variable de g. S'il existe un unificateur principal σ tel que

1) $\sigma t = \sigma l$

2) $\sigma r \not\geq \sigma l$

3) $\sigma d \not\geq \sigma g$

alors $\langle \sigma d, \sigma(g[r]) \rangle$ est une **paire critique généralisée**. Le processus de génération des paires critiques genéralisée est appelé **superposition**

généralisée.

○

Le lecteur remarquera que dans le cas où g=d et l=r sont orientables, (donc lorsque nous avons g→d et l→r) nous retrouvons la définition classique des paires critiques, en effet les conditions 2) et 3) deviennent

2') σl > σr

3') σg > σd

Nous donnons maintenant une extension de la notion de réduction qui consiste à utiliser les instances orientables d'équations comme des règles de réécriture.

Définition 6.3 -- Un terme s[t] est **réductible par** une équation l=r s'il existe une substitution σ telle que

1) σl = t

2) σl > σr

On dit aussi que l'équation l=r **réduit** le terme s[t] (c'est-à-dire s[σl]) en le terme s[σr].

○

Nous retrouvons la définition de la réduction classique lorsque l=r est orientable en l→r car dans ce cas la condition 2) est automatiquement vérifiée.

Rappelons que l'ordre > considéré est bien-fondé, et donc qu'aucun terme ne peut être réduit indéfiniment.

Exemple 6.1 -- Considérant l'équation x*y=y*x, et un ordre > qui peut orienter a*b > b*a (par exemple l'ordre recursif de décomposition avec a > b et un statut gauche-droite pour *), le terme a*b est réductible par x*y=y*x en b*a, mais b*a n'est pas réductibe puisqu'on n'a pas b*a > a*b. De plus, si u et v sont des variables, u*v ne peut pas être réductible puisqu'on ne peut pas avoir u*v > v*u (ni d'ailleurs v*u > u*v). Cette équation ne peut donc pas être utilisée pour réduire des termes non clos.

∇

6.2.3. Règles d'inférence

Les règles d'inférence de UKB sont très similaires à celles de SKB. Les réductions sont remplacées par les réductions généralisées et les paires critiques par les paires critiques généralisées (les superpositions sont des superpositions généralisées).

Notons que contrairement à SKB, une équation non orientée est utilisée pour la simplification par réécriture généralisée (utilisant les notations du paragraphe 5.2.1, une règle de S est utilisée comme simplificateur, par la réduction généralisée, avant d'aller dans E).

Nous proposons, sous forme intuitive, les règles d'inférence suivantes qui généralise celles de SKB:

U1▶ La réduction par les simplificateurs de S (règles orientées ou non), des règles de R, E et S. Ce mécanisme peut engendrer de nouveaux simplificateurs à partir des règles ou équations simplifiées.

U2▶ Une règle de S (orientée ou non) qui ne simplifie plus est transférée dans E.

U3▶ Le calcul d'une paire critique généralisée de E avec une équation ou une règle de R.

U4▶ Le transfert dans R d'une règle (orientée ou non) de E dont les paires critiques avec R ont toutes été calculées.

U5▶ L'élimination des équations qui sont subsumées par d'autres (s'il exsite σ telle que σl=g et σr=d alors g=d est éliminée)

6.2.4. Algorithme UKB

Les règles d'inférence de UKB sont les mêmes que SKB (S1,..,S6) en remplaçant la superposition par la superposition généralisée et la réduction par la réduction généralisée. Nous avons de plus la règle de subsomption suivante

U7.1 $$\frac{E,RU\{l=r\},SU\{g=d\}}{E,RU\{l=r\},S}$$ • ∃ σ / σl=g et σr=d ou σl=d et σr=g

U7.2 $$\frac{EU\{l=r\},R,SU\{g=d\}}{EU\{l=r\},R,S}$$ • ∃ σ / σl=g et σr=d ou σl=d et σr=g

Appelons ces règles d'inférence U1) U2).. U7).

La preuve de correction se fait de la même manière que celle de SKB, en utilisant le formalisme de Bachmair-Dershowitz-Hsiang et la théorie UKB.

Nous appelons une procédure UKB toute procédure utilisant les règles d'inférence Ui.

Considérant un ensemble d'équations E, et un ordre < de simplification fort. Trois éventualités peuvent se produire lorsque l'algorithme UKB est appliqué à E et <:

1) Il n'y a plus de paires critiques divergentes et UKB s'arrête avec un ensemble R de règles toutes orientées. Dans ce cas le système de règles trouvées est convergent, et donc tous les termes E-égaux ont la même forme normale dans R

2) Il n'y a plus de paire critique divergente et UKB s'arrête avec un ensemble R de règles qui ne sont pas toutes orientées. Dans ce cas R est un système convergent sur les termes clos et UKB peut néanmoins servir de procédure de décision pour le problème du mot dans E (paragraphe 5.3)

3) La superposition généralisée engendre toujours des paires critiques non convergentes, UKB ne termine pas. Dans ce cas étant donné deux termes clos E-égaux, il existe un système Ri de règles et d'équations trouvées à une étape i de l'algorithme, pour lequel ces termes ont la même forme normale.

Nous donnons deux exemples où le résultat de UKB est un système convergent sur les termes clos. Ces systèmes convergents sur les termes clos vont servir à prouver des théorèmes équationnels comme on le verra dans le paragraphe 5.3.

Exemple 6.2 -- considérant les deux équations suivantes:

$$(x.y).(z.w) = (x.z).(y.w)$$
$$(x.y).x = x$$

Le premier axiome est permutatif, il ne peut donc pas être orienté, ce qui met en échec l'algorithme de complétion de Knuth-Bendix. Avec la procédure de complétion UKB, nous obtenons le système suivant

$$(x.y).x \rightarrow x$$
$$x.(y.z) \rightarrow x.z$$
$$(x.y).z = (x.w).z$$
$$((x.y).z).w \rightarrow x.w$$

Ce système est convergent sur les termes clos.

Exemple 6.3 -- Considérons l'ensemble suivant d'équations

$$(x.y).x = x$$
$$x.(y.z) = x.z$$
$$(x.y).z = f(x,z)$$
$$f(x,z).w = x.w$$
$$((x.y).z).w = x.w$$

UKB s'arrête avec le système suivant

$$f(x, x) \rightarrow x$$
$$x \,.\, (y \,.\, z) \rightarrow x \,.\, z$$
$$(x \,.\, y) \,.\, z \rightarrow f(x, z)$$
$$f(x, z) \,.\, w \rightarrow x \,.\, w$$
$$f(x \,.\, y, w) = x \,.\, w$$
$$f(x, y \,.\, z) \rightarrow f(x, z)$$
$$y \,.\, f(x, z) \rightarrow y \,.\, z$$
$$f(f(x, z), y) \rightarrow f(x, y)$$
$$f(x, f(y, z)) \rightarrow f(x, z)$$
$$f(x \,.\, z, w) = f(x \,.\, y, w)$$

6.2.5. Implantation de UKB

La procédure donnée dans ce paragraphe est une réalisation de la procédure UKB. Le schéma général de la procédure UKB est donc le même que SKB, nous utilisons les paires critiques généralisées et la réduction généralisée.

```
% l'initialisation: R = ø, E = ø, S = l'ensemble des équations
% de l'utilisateur.

UKB = proc(R,E,S: ensemble de règles)
   USIMP(S,E,R)
   % à la sortie de USIMP, S est vide
   répéter jusqu'à E = ø

      ● Prendre la première règle l→r (orientée ou non) de E (E:=E-{l→r})
      ● Engendrer UNE paire critique non triviale normalisée g=d
        de l→r sur RU{l→r}
        • si g=d est subsumée par une équation de R ou de E alors continue
        • USIMP({g→d}US,E,R)
        • Si l→r est simplifiée alors stopper la génération
                                    des paires critiques
        • Si il n'y a plus de paires critiques pour l→r alors R:=RU{l→r}
fin UKB



USIMP = proc(S,E,R: ensemble de règles)
   Tant que S ≠ ø faire
      • Prendre la première règle (orienter ou non) l=r de S
      • USIMPLF({l→r},S)
      • USIMPLF({l→r},E)
      • USIMPLF({l→r},R)
      • E:=EU{l→r}
fin USIMP



USIMPLF = proc({l→r},T: ensemble de règles)
   répéter jusqu'à ce que toutes les règles de T aient été considérées.
      • Prendre une règle g→d de T
      • Si g→d est simplifiée par l→r en g'→d' alors
        S:=SU{g'=d'}
        T:=T-{g→d}
fin USIMPLF
```

Remarque 6.1 -- Il est nécessaire d'utiliser un test de subsomption afin de détecter et d'éliminer les paires critiques qui sont instances des équations du système. Cela pour éviter à l'algorithme de reproduire indéfiniment les mêmes équations non-orientables. Cette condition n'est pas nécessaire dans l'algorithme de complétion de Knuth-Bendix puisqu'on

suppose que toutes les paires critiques sont orientables.

Les règles sont considérées dans une h-table (table d'adressage dispersé) comme pour le cas de SKB (table h-codée par le symbole de tête du membre gauche l d'une règle l→r). Dans le cas de UKB, les équations sont utilisées dans la réduction, l'ensemble de ces équations est aussi représenté par une table h-codée. Si on utilise l'équation l=r pour réduire, il est nécessaire de considérer aussi l'équation r=l. Une insertion dans la table h-codée utilisant les symbole de tête de r et de l permet de retourner l=r ou r=l suivant que le symboles de tête du terme à réduire est le même que celui de l ou de r.

6.3. Utilisation de UKB dans les preuves équationnelles

Nous présentons dans ce paragraphe une méthode pour décider du problème du mot dans les théories équationnelles utilisant UKB.

Dans le cas où UKB retourne un système convergent sur les termes clos, la procédure pour décider de problème du mot dans une théorie équationnelle E donnée au départ est la suivante: étant donnée une équation s=t, nous considérons sa négation, et la skolémisons pour obtenir $\hat{s} \neq \hat{t}$: toutes les variables de $\hat{s}$ et $\hat{t}$ sont remplacées par des constantes de Skolem et donc admettent des formes normales uniques dans le système convergent sur les termes clos considéré. La forme normale de $\hat{s}$ est égale à la forme normale de $\hat{t}$ si et seulement si, d'après le théorème de Hsiang et Rusinowitch, l'équation est un théorème de E.

UKB est une méthode de semi-décision pour le problème du mot dans les théories équationnelles comme le montre le théorème suivant.

Théorème 6.1 -- [Hsiang et Rusinowitch]-- Soit E une théorie équationnelle et s=t une équation. s=t est un théorème dans E si et seulement si la procédure UKB, appliqué à EU{ŝ ≠ t̂}, produit une contradiction ~~a≠a~~.

La démonstration est donnée dans [Hsiang-Rusinowitch-85b], elle utilise une technique sur les arbres sémantiques transfinis [Hsiang-Rusinowitch-85a].

Ce théorème nous assure la preuve de s=t par réduction de ŝ ≠ t̂ en utilisant les règles et les équations de E.

On peut faire un parallèle entre ce théorème et la formulation réfutationelle du résultat de Huet [Huet-81]: étant donnée une théorie équationnelle E et une équation s=t, s=t est un théorème de E si et seulement si la procédure de complétion de Knuth-Bendix engendre suffisamment de règles pour réduire l'inéquation ŝ ≠ t̂ en a ≠ a. Il est cependant nécessaire que l'algorithme de Knuth-Bendix n'engendre pas d'équations non orientables. Cela n'est plus nécessaire pour le cas de UKB, puisque les équations servent aussi à réduire ŝ ≠ t̂.

Il n'est pas nécessaire non plus de trouver un système convergent sur les termes clos pour réfuter une inéquation, on peut le faire "à la volée" c'est-à-dire pour chaque nouvelle règle engendré par UKB, tester si elle réduit l'inéquation à la contradiction a ≠ a.

Exemple 6.4 -- Considérons la spécification du IF qui ne peut pas être orientée par la procédure classique de complétion à cause des équations permutatives qu'elle contient.

```
if(true, x, y) == x
if(false, x,y) == y
if(x, x, y) == if(x, true, y)
if(x,y,x) == if(x,y,false)
if(botom, x,y) == botom
if(x, botom, botom) == botom
if(xp, if(xp, x,y),z) == if(xp,x,z)
if(xp,x,if(xp,y,z)) == if(xp,x,z)
if(xp,if(xq,x,y),if(xq,u,v)) == if(xq,if(xp,x,u),if(xp,y,v))
if(if(xp,x,y),u,v)== if(xp,if(x,u,v),if(y,u,v))
```

Le but est de montrer l'équation suivante

$$if(x, u, if(y, u, v)) == if(if(x, true, y), u, v)$$

Nous la skolémisons et nous considérons sa négation:

$$if(px, pu, if(py, pu, pv)) \neq if(if(px, true, py), pu, pv)$$

où px, pu, py, pv sont des nouvelles constantes de Skolem. Considérant cet ensemble d'équation et l'inéquation à réfuter, UKB arrive à une contradiction (les deux membres réduits de l'inéquations sont unifiable), en 15 secondes, ce qui achève la preuve sans avoir besoin de compléter le système initial.

∇

Exemple 6.5 -- Considérons la spécification suivante:

```
   x+y == x+x
(x-y)+z == (x+z)-y
(x+y)-y ==x
```

et l'inéquation suivante

$$(a-b)+c \neq a$$

en considérant l'ordre suivant c>->a>b nous avons une contradition en 13 secondes.

∇

Hsiang et Rusiniwitch [Hsiang-Rusinowitch-85b] montrent que cette méthode de preuve peut être généralisé au cas ou on a plusieurs inéquations et où ces inéquations contiennent des variables, plus précisement dans le cas des théories équationnelles contenant un ensemble fini d'équations et d'inéquations avec toutes les variables quantifiées universellement. Pour cela on définie la superposition d'une équation et d'une inéquation comme suit:

Définition 6.4 -- Soit l=r une équation et g[t]≠d une inéquation, s'il existe σ une substitution telle que

1) σt = σl

2) σr ≱ σl

alors σ(g[r]≠d) est une inéquation engendrée par superposition de g[t]≠d et l=r.

O

La contradiction est engendrée lorsqu'on a une inéquation g≠d avec σg=σd.

Les inéquations peuvent être considérées comme des équations et ajoutées à l'ensemble des équations de la théorie considérée (ŝ ≠ t̂=1 où ≠ est un nouveau symbole de fonction) et l'équation x≠x=0; l'équation est réfutée lorsque la contradiction 1=0 est engendrée. Cette contradiction ne

peut jamais être engendrée par superposition de deux inéquations (équations de symbole de tête ≠), il est donc inutile de superposer de telles inéquations. Pour cette raison, dans l'implantation SBREVE, l'ensemble des inéquations est stocké à part, dans un ensemble N. Les paires critiques engendrées par la superposition d'une inéquation de N et d'une règle de R ou de E est une inéquation, elle est donc mise dans N si ce n'est pas une contradiction.

Exemple 6.6 -- Soit la spécification suivante d'un problème donné dans "5th Assoc. of Automated Reasoning Newsletter, 1985":

```
s(x, s(y, z)) == s(f(x, y), z)
s(m, x) == s(x, x)
s(a,x) ≠ x
```

Le système SBREVE arrive à la contradiction suivante en 5 secondes sur VAX 785.

```
s(m,f(x,m)) ≠ s(m,f(x,m))
```

∇

6.4. Conclusion

UKB est une méthode de complétion qui n'échoue pas à cause d'équations non orientables. Nous avons vu au chapitre précédent une autre méthode, la réécriture équationnelle, pour résoudre le même problème. Pour cette dernière méthode (réécriture équationnelle) nous avons besoin d'algorithmes d'unification et de filtrage pour la théorie considérée alors que pour UKB, on utilise l'unification dans la théorie vide. Dans certains cas contenant

les axiomes d'associativité et de commutativité, on peut ne pas trouver de système convergent fini par UKB (l'algorithme ne termine pas en engendrant indéfiniment des nouvelles paires critiques), alors que la réécriture équationnelle termine et donne un système convergent fini modulo AC. Une approche naturelle serait de combiner ces deux méthodes.

PARTIE II

7. Egalité modulo un ensemble d'axiomes E

7. Egalité modulo un ensemble d'axiomes E

Pour décider de l'égalité de deux termes modulo un ensemble d'axiomes E, il n'est pas toujours possible d'utiliser les techniques de complétion de Knuth-Bendix sur l'ensemble d'équations E. De plus, dans la méthode de complétion équationnelle, E est justement un ensemble d'équations où ces techniques échouent. Des axiomes, tels la commutativité, ne peuvent pas être utilisés comme des règles orientées de gauche à droite, puisque la propriété de terminaison n'est pas satisfaite. Ces techniques ne sont pas possibles dans le cas, comme celui de l'ensemble E des axiomes d'associativité et d'idempotence, où l'ensemble des règles engendrées est infini.

Pour résoudre ce problème, nous exploitons une technique très souvent utilisée en programmation, et nous la formalisons. Cette technique consiste à utiliser des structures de données associées à l'ensemble des axiomes E, pour décider de l'égalité modulo E. Ainsi ces structures de données permettent de représenter les classes d'équivalence des termes modulo l'ensemble d'axiomes E en question.

Cette démarche est naturelle, puisqu'une structure de données peut être vue comme une espèce de structures [Bourbaki-68]. Les ensembles d'axiomes ne sont qu'une description de cette structure, comme l'explique A. Joyal [Joyal-81]:

Il existe déja un concept précis d'espèce de structures. La description d'une espèce particulière se fait souvent en spécifiant les conditions que doit satisfaire une structure pour appartenir à cette espèce. Cette description peut prendre la forme d'une axiomatisation de l'espèce considérée.

Il est donc naturel d'associer à un ensemble d'axiomes une structure, ce n'est que voir la même chose de deux points de vue différents. Nous gardons l'aspect ensemble d'axiomes lorsque nous manipulons des termes de l'algèbre libre, et nous considérons la structure lorsque cela est nécessaire.

Notation -- Soient **F** l'ensemble des symboles de fonction, et M(F,X) l'algèbre libre engendrée par X. F_e désigne l'ensemble des symboles de fonction qui vérifient les axiomes de l'ensemble E, F_{ac} sera l'ensemble des symboles de fonction associatifs et commutatifs et F_i celui des symboles idempotents.

7.1. Introduction de la méthode sur un cas particulier

Dans ce paragraphe nous introduisons une méthode de décision pour l'égalité modulo l'ensemble d'axiomes d'associativité et de commutativité. Nous définissons un ensemble appelé **FNAC** (AC pour l'ensemble des axiomes utilisés, dans ce cas l'axiome d'associativité et de commutativité) de structures récursives notées FN_{ac}, et nous montrons que $M(F,X)/=_{AC}$ (l'ensemble des termes quotienté par congruence $=_{AC}$) est en bijection avec

une partie de FNAC. Nous définissons ensuite dans FNAC un système de réécriture dans lequel la forme normale d'un terme t (ou plus exactement de son image dans FNAC) est le représentant de la classe d'équivalence de tous les termes égaux à t modulo les axiomes d'associativité et de commutativité.

7.1.1. Algèbre de Structures

Soit F_{ac} l'ensemble des symboles AC, X l'ensemble des variables et $M(F_{ac},X)$ la F-algèbre libre sur X.

Définition 7.1 -- FN_{ac} est par définition

* soit une variable,
* soit un couple formé d'un symbole de fonction et d'un multiensemble de FN_{ac}.

$$FN_{ac} ::= \text{variable} \mid (\text{symbole de fonction AC}, \{\{ FN_{ac} \}\})$$

où {{x}} est le multiensemble qui contient x.

○

On peut munir l'ensemble des FN_{ac} d'une structure de F-algèbre, que l'on note FNAC en définissant les opérations f_{ac} suivantes :

$$\forall f \in F,\ ar(f) = k,$$

$$f_{ac} : FNAC^k \rightarrow FNAC$$

$$t_1,..,t_k \mapsto f_{ac}(t_1,..,t_k)=(f, \{\{t_1,..,t_k\}\})$$

M(F,X) peut être considérée comme une partie de l'algèbre FNAC, et

ceci en considérant l'image de M(F,X) par l'application suivante :

I_{ac} : M(F,X) → FNAC

$\gamma \mapsto$ si γ est une variable alors γ

sinon $\gamma = f(t_1,..t_n)$

$(f, \{\{I_{ac}(t_1)..,I_{ac}(t_n)\}\})$

Un élément de FNAC est l'image d'un terme par l'application I_{ac} s'il vérifie les contraintes d'arité correspondantes.

Exemple 7.1 -- Un terme f(f(x,y),z) peut être considéré comme l'élément (f,{{(f,{{x,y}}),z}}) de FNAC.

∇

7.1.2. Système de Réécriture de Structures

Nous définissons maintenant un système de réécriture dans l'ensemble FNAC; ce système sert à caractériser la classe des termes égaux à un terme t modulo les axiomes d'associativité et de commutativité. Cette classe est caractérisée par la forme normale de $I_{ac}(t)$ dans ce système de réécriture. En d'autres termes le quotient de l'algèbre M(F,X) par la congruence $=_{AC}$ est en bijection avec une partie de FNAC.

Notation -- Pour simplifier les notations , $(f,\{\{x_1,..,x_n\}\})$ est noté $f\{\{x_1,..,x_n\}\}$.

Soit R[AC] le système de réécriture constitué des règles suivantes :

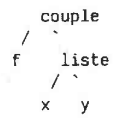
Schéma 1: $f\{\{x_1,..x_n,f\{\{y_1,..,yn\}\}\}\} \rightarrow f\{\{x_1,..,x_n,y_1,..,yn\}\}$

Schéma 2: $f\{\{x\}\} \rightarrow x$

Remarquons que le système précédent, tel qu'il est présenté n'est pas un système de réécriture des termes mais un système de réécriture de structures. Néanmoins il peut être considéré comme un système de réécriture de termes puisque les structures FN_{ac} elles mêmes peuvent être considérées comme des termes d'une certaine F-algèbre, moyennant la donnée d'une précédence sur l'ensemble des symboles de fonction et des variables. Par exemple la structure $f\{\{y,x,f\{\{x\}\}\}\}$ est le terme $t=f_1(x,y,f_2(z))$ de la F-algèbre où f_1 et f_2 appartiennent à F et d'arité respective 3 et 1, avec en plus la précédence x<y<f2.

Remarquons que l'on peut aussi considérer ces structures comme des termes de symboles de fonction d'arité variable, le même terme t sera alors $\overline{f}(x,y,\overline{f}(z))$ où $\overline{f}$ est un symbole de fonction d'arité variable.

Une autre manière de considérer les structures FN_{ac} comme des termes est la suivante : une structure f{{x,y}} est le terme suivant



Où "liste" et "couple" sont deux nouveaux symboles de fonction.

La justification de la considération des structures en tant que termes étant faite, nous revenons aux structures FN_{ac} que nous manipulons alors comme des termes.

Lemme 7.1 -- Le système de réécriture R[AC] est convergent.

Preuve :

la terminaison est due à la taille décroissante des termes. Nous laissons au lecteur le soin de vérifier qu'il n'y a pas de paires critiques entre ces deux règles.

□

Le système convergent R[AC] va donc nous servire à caractériser les classes des termes égaux modulo les axiomes d'associativité et de commutativité. C'est l'objet du paragraphe suivant.

7.1.3. Caractérisation des Classes Modulo AC

Définition 7.2 -- Soit M un terme, on appelle forme normale associative-commutative, et on la note $FN_{ac}(M)$, la forme normale de $I_{ac}(M)$ dans R[AC].

○

Proposition 7.1 -- Soient M et N deux termes,

$$M =_{AC} N \Leftrightarrow FN_{ac}(M) = FN_{ac}(N)$$

Preuve :

Deux points à démontrer :

1) R[AC] contient AC (trivial) et

2) $t_1 \longleftrightarrow^*_{R[AC]} t_2$ ssi $t_1{\downarrow}_{R[AC]} = t_2{\downarrow}_{R[AC]}$, d'après la canonicité de R[AC].

□

Cette méthode nous a donc permis de ramener l'égalité modulo les axiomes d'associativité et de commutativité à une égalité sur les multiensembles, puisque la structure FN_{ac} d'un terme M est un couple composé d'un symbole de fonction et d'un multiensemble.

Exemple 7.2 -- Pour $f \in F_{ac}$, les deux termes suivants ont la même forme normale AC f{{x,y,a,b}}

▽

7.1.4. Réalisation

La mise en pratique de cette méthode pour décider de l'égalité modulo AC est simple puisque l'implantation de la structure FN_{ac} ne pose pas de problèmes particuliers. En utilisant des langages de programmation basés sur les types abstraits, tel CLU [Liskov,etal.-81], il suffit de définir le type multiensemble. Quant au système de réécriture R[AC], il peut être réalisé par une fonction manipulant les structures FN_{ac}. L'égalité modulo AC devient de cette manière une simple égalité syntaxique sur les structures FN_{ac}. Nous donnons dans le paragraphe 7.2.4 une procédure permettant

la réalisation de cette méthode. En fait nous donnons une procédure générale dont, la méthode présentée dans ce paragraphe est un cas particulier.

7.2. Et pour les Autres Ensembles d'Axiomes?

Dans ce paragraphe nous caractérisons l'égalité dans la théorie équationnelle où les symboles de fonction vérifient l'axiome E. nous montrons comment l'égalité modulo cet ensemble d'axiomes E, est caractérisée en associant à chaque ensemble E une structure adaptée. Par exemple pour l'axiome de commutativité C, nous utilisons la structure des couples permutatifs.

Le paragraphe suivant est une présentation des structures de données utilisées dans la suite pour les axiomes d'associativité, de commutativité, d'idempotence ou de toute combinaison de ces trois axiomes.

7.2.1. Structures de données pour A, C ou I.

Soit E un ensemble d'axiomes parmi les sept ensembles suivants :

A
C
I
AC
AI
CI
ACI

A chaque ensemble d'axiomes E on fait correspondre une structure de

donnée str_E. Commençons par les définir.

Trois opérations sont nécessaires pour définir str_E :

$$nil_E : \rightarrow str_E$$

$$add_E : M(F,X) \times str_E \rightarrow str_E$$

$$^\circ : str_E \times str_E \rightarrow str_E$$

Dans le tableau suivant nous décrivons ces opérations et les relations entre elles.

E	str_E	$add_E(x,nil_E)$	propriétés
A	liste	/x/	add(x,11)°12 = add(x,11°12)
C	couple permutatif	[x]	[x]°[y] = [x,y] = [y,x]
I	couple sans répétition	(x)	(x)°(y) = (x,y) add(x, (x)) = (x)
AC	multi-ensemble	{{x}}	add(x, M_1)°M_2 = add(x, M_1°M_2)
AI	liste sans répétition immédiate	//x//	//x//°//y// = si x = y alors //x// sinon //xy//
CI	couple permutatif sans répétition	[[x]]	[[x]]°[[y]] = si x = y alors [[x]] sinon [[x,y]] = [[y,x]]
ACI	ensemble	{x}	

Nous utilisons la notation str_E pour désigner aussi l'application qui à tout objet t fait correspondre un élément de la catégorie des str_E, par

exemple si t est un arbre, $str_{ACI}(t) = \{t\}$ qui est l'ensemble qui contient l'arbre t, et $str_A(t) = /t/$ la liste contenant un seul élément t.
Si $t \in M(F,X)$ alors

$$str_E(t) = add_E(t, nil_E)$$

7.2.2. Forme normale modulo E

Soit FN_e une structure récursive définie, pour E quelconque par :

$$FN_e ::= variable \mid (f, str_E(FN_e))$$

Pour E fixé FN_e est la structure récursive associée à la structure de données str_E. Par exemple FN_{AC} est la structure FN_e pour E = AC associée à la structure str_{AC} qui est la structure de multiensemble.

L'ensemble des FN_e est appelé FNE.

FNE peut être muni d'une structure de F-algèbre de la même manière que FNAC dans le paragraphe précédent.

M(F,X) peut être aussi vue comme une partie de FNE (ou isomorphe à une partie de FNE)

L'application qui à un terme M fait correspondre son image dans FNE est appelée I_E. Nous avons certainement besoin d'un test sur le symbole de tête du terme afin de déterminer la théorie équationnelle E à laquelle il appartient, et donc déterminer la structure de données str_E nécessaire pour

définir FN_e. Nous introduisons pour celà l'application **TH** qui à tout symbole de fonction fait correspondre la théorie à laquelle il appartient.

Exemple 7.3 -- Soient $f \in F_{ac}$, a et b des constantes.

$$\begin{aligned} I_{TH[f]}(f(x,f(a,b))) &= I_{AC}(f(x,f(a.b))) \\ &= (f, str_{AC}(I_{AC}(x)) \circ (I_{AC}(f(a,b))) \\ &= (f, \{\{x\}\} \cup \{\{(f, \{\{x, y\}\})\}\} \\ &= (f, \{\{x, (f, \{\{x, y\}\})\}\}) \end{aligned}$$

∇

Notation -- Pour simplifier les notations, le couple $(f, str_E(t))$ sera noté $fstr_E(t)$. Ainsi le terme de l'exemple précédent est noté f{{x,g[x,y]}}

Pour chaque ensemble d'axiomes E, et donc pour chaque ensemble FNE nous définissons un système de réécriture R[E] contenant deux règles $R_A[E]$ et $R_I[E]$, telle que pour $S \in \{A,I\}$, si $S \in E$ alors la règle $R_S[E]$ appartient à R[E].

$R_I[E]$ $\quad (f, str_{TH(f)}(t)) \rightarrow t$

$R_A[E]$ $\quad (f, str_{TH(f)}(x_1,..x_k, (f, str_{TH(f)}(y_1,..y_k), z_1,..z_k)$

$$\rightarrow (f, str_{TH(f)}(x_1,..,x_k) \circ str_{TH(f)}(y_1,..,y_k) \circ str_{TH(f)}(z_1,..,z_k))$$

Tous les systèmes R[E] sont convergents à cause de l'absence des paires critiques et de la décroissance de la taille des membres droits.

La forme normale d'un terme M (ou plus précisement de $I_E(M)$) dans le système de réécriture R[E] nous permet de caractériser la classe

d'équivalence de tous les termes égaux à t modulo E.

Définition 7.3 -- Soit M un terme, on appelle E-forme normale de M, ce qu'on note $FN_E(M)$, la forme normale de $I_E(M)$ dans le système de réécriture R[E].

○

Exemple 7.4 -- La forme AC-normale du terme t = f(x,f(a,b)) de l'exemple précédent est la la forme normale de $I_{AC}(t)$ dans le système $R_A[AC]$. nous avons (exemple précédent)

$$I_{AC}(t) = (f, \{\{x, (f, \{\{a, b\}\}\}\}))$$

en appliquant la règle $R_A[AC]$ sur $I_{AC}(t)$, nous obtenons

$$(f, \{\{x\}\cup\{\{a,b\}\}) =$$
$$(f, \{\{x,a,b\}\})$$

d'où

$$FN_{AC}(f(x,f(a,b))) = (f, \{\{x,a,b\}\})$$

∇

La caractérisation des classes d'équivalence est donnée par la proposition suivante.

Proposition 7.2 -- Soient deux termes M et N, et E un ensemble d'axiomes parmis les 7 ensembles définis au paragraphe 7.2.1, M et N sont égaux modulo E si et seulement si $FN_E(M) = FN_E(N)$.

Preuve :

D'après la canonicité du système de réécriture R[E] considéré.

□

Une implantation de cette méthode est donnée dans le paragraphe 7.2.4.

7.2.3. Forme normale dans un mélange de structures

Nous considérons dans ce paragraphe une théorie équationnelle où certains symboles de fonction satisfont un ou plusieurs axiomes parmi A, C ou I. Les symboles de fonction satisfont donc un ensemble d'axiomes parmi les sept ensembles donnés au paragraphe 7.2.1. Un terme dans cette théorie peut donc contenir des symboles de fonction A, d'autres AC, etc.. Pour chaque symbole de fonction f nous avons une théorie E et une structure de donnée associée str_E.

Les différents systèmes définis dans le paragraphe précédent nous permettent de définir une forme normale des termes, utilisant les différentes structures de données définies au le paragraphe 7.2.1.

Définition 7.4 -- Soit un terme $M=f(..t_i..)$. La forme normale de M, ce qu'on note FN(M) est

$$FN_{TH(f)}str_{TH(f)}(FN(t_1)^\circ..^\circ str_{TH(f)}(FN(t_i))^\circ..)$$

○

Exemple 7.5 -- Considérons le terme t = f(x,f(a,g(y,z))) lorsque f est AC et g est CI.

$$
\begin{aligned}
FN(t) &= FN_{TH(f)} f str_{TH(f)}(FN(x)) \circ str_{TH(f)}(FN(f(a,f(y,z))) \\
&= FN_{AC} f\{\{FN(x)\}\} \cup \{\{FN(f(a,g(y,z)))\}\} \\
&= \ldots \\
&= FN_{AC} f\{\{x\}\} \cup \{\{(FN_{AC}(f\{\{a,FN_C(g(x,y)))\}\} \\
&= \ldots \\
&= FN_{AC} f\{\{x,(f\{\{a,g[[x,y]]\}) \\
&= f\{\{x,a,g[[x,y]]\}\}
\end{aligned}
$$

∇

Proposition 7.3 -- Soient deux termes M et N; M et N sont égaux modulo l'ensemble des axiomes vérifiés par leurs symboles de fonction si et seulement si FN(M) = FN(N).

Exemple 7.6 -- Le terme de l'exemple précédent est égal au terme f(x,g(y,z)), puisqu'ils ont la même forme normale.

7.2.4. Réalisation de la forme normale dans un mélange de théories

La caractérisation qu'on vient de présenter dans le paragraphe précédent peut être réalisée facilement. Nous donnons dans ce paragraphe une réalisation pour le cas de mélange de théories, implantée dans le système REVE 3 pour le cas AC et C. L'implantation de la méthode pour le cas d'une seule théorie E est un cas particulier de cette implantation.

Le calcul de la forme normale d'un terme est donné sous la forme d'une procédure appelée FN. La procédure FN prend en entrée un terme et retourne sa forme normale.

```
FN(M) = proc (M: terme) retourne (forme normale de M)
      si ( M = cte ou variable ) alors retourner( M, nil )
      sinon M = f(t1, .., tn)
            si f vérifie lensemble d'axiome E tq A ∈ E alors
                  retourner( APLATIR( f, strE(FN(t1)), strE(FN(t2))))
            sinon % f vérifie les axiomes de E
                  retourner(f, strE(FN(t1))°strEFN(t2))
            fsi
      fsi
      fFN
```

f vérifiant les axiomes de E dont A

```
APLATIR = proc (f: top, (a1, b1), (a2, b2): (top, str)) retourne ((top, str))
      NV := nilE
      pour i:= 1→2 faire
            si ai = f alors NV := NV°strE(bi)
            sinon NV := NV°strE((ai,bi))
            fsi
      fpour
      retourner(f, NV)
      fAPLATIR
```

Dans ce paragraphe nous avons caractérisé la E-égalité en associant une structure de donnée à E et en caractérisant l'égalité de telle structures. Nous avons ramené l'égalité une égalité modulo E à une égalité syntaxique sur les structures de donnée. Pour la théorie équationnelle où chaque symbole de fonction f_k vérifie la théorie E_k, le problème d'égalité dans cette théorie est ramené à l'égalité des structure str_{Ek} correspondantes. Une procédure de construction des structures pour le cas particulier où les symboles de fonction considérés vérifies un ou plusieur axiomes parmi A, C et I est donné. Nous n'avons pas étudié le problème de la correspondance systématiquement d'une structure de donnée à un ensemble d'axiomes.

8. **Etude algébrique du problème du filtrage**

8. Etude Algébrique du Probleme de Filtrage

8.1. Introduction

Dans ce chapitre nous étudions le problème de filtrage dans les théories équationnelles d'un point de vue algébrique. Nous résolvons des équations et des systèmes d'équations. Nous définissons des opérations élémentaires sur les équations et les systèmes d'équations. Ces transformations appliquées à l'équation (pour le problème de filtrage) M«N, permettent de trouver l'ensemble de filtres de M vers N.

Plus précisement, étant donnés deux termes M et N, chercher l'ensemble des filtres de M vers N dans la théorie équationnelle E (ou chercher les E-solutions), revient à transformer l'équation $M \ll_E N$ en un système d'équations (ou plusieurs systèmes) plus simples, ayant le même ensemble de solutions que l'équation M«N. Ce processus est itéré jusqu'à l'obtention d'un ensemble de systèmes d'équations triviales, auxquelles correspondent les substitutions cherchées.

Pour résoudre une équation, nous la transformons en équations plus simples. Grâce à trois types de transformations, appelés **décomposition**, **mutation** et **fusion** .

Nous prouvons de la complétude de la méthode, en montrant que les transformations appliquées sont des **E-équivalences** dans le sens où

l'ensemble des E-solutions d'un système donné d'équations est le même que l'ensemble des E-solutions du système transformé.

Une manière duale de résoudre un système d'équations dans une théorie équationnelle E, consiste à transformer le système en un ensemble de systèmes équivalents à résoudre dans des théories équationnelles plus simples que E, par exemple des sous-ensembles de E. Cette méthode est l'objet du chapitre: Union de Théories. Elle fut appliquée pour la première fois aux problèmes d'unification (ou résolution d'équations pour le problème d'unification) par C. Kirchner [Kirchner-85] et K. Yelick [Yelick-85] puis par E. Tiden [Tiden-86] et A. Herold [Herold-86]. Cette méthode ramène l'étude d'un problème dans une théorie E à celui des problèmes dans les théories E_i telles que $\bigcup_i E_i = E$, d'où le nom du chapitre correspondant.

8.2. Substitutions

Dans ce paragraphe nous étudions les opérations sur les substitutions.

Définition 8.1 -- Soit E un ensemble d'axiomes. Nous dirons qu'un ensemble de substitutions σ_i pour $i \in [1,p]$ est **inconsistant** modulo E, s'il existe j et k appartenant à [1,p], et s'il existe x qui appartient à la fois à $D(\sigma_j)$ et $D(\sigma_k)$ tel que $\sigma_j(x) \neq \sigma_k(x)$ modulo E. Nous disons que la suite des substitutions est **consistante** dans le cas contraire.

○

Définition 8.2 -- Soit σ_i pour $i \in I \subseteq \mathbb{N}$ un ensemble de substitutions consistante, alors par définition la substitution

$$\sigma = \sigma_1 \wedge \sigma_2 \wedge \ldots \wedge \sigma_k$$

est la substitution de domaine $\bigcup_i D(\sigma_i)$, définie par

$$\sigma(x) = \text{si } x \in D_i \text{ alors } \sigma_{i|Di}(x)$$

L'opération $\wedge$ est appelé **conjonction**. Dans le cas d'une suite inconsistante de substitutions, le résultat de l'opération de conjonction $\wedge$ est par définition ø, la substitution résultat n'existant pas.

○

Cette opération correspond à la composition des substitutions à un seul niveau, c'est tout ce dont on a besoin dans un problème de filtrage. Par exemple le filtre de f(x,y) vers f(y,a) est {x←y,y←a} et non pas {x←y}. Cela provient du fait que seules les variable de gauche sont instantiées (σM=N). Nous revenons plus en détail sur ce problème dans les paragraphes suivants.

Nous généralisons l'opération $\wedge$ aux ensembles de substitutions.

Définition 8.3 -- Soient S_i (pour $i \in [1,k]$) une suite d'ensembles de substitutions.

$$S_i = \{\sigma^i_j\} \text{ avec } i \in I \text{ et } j \in J_i$$

alors $S_1 \wedge S_2 \wedge .. S_k$ est un ensemble de substitutions défini par :

$$S_1 \wedge S_2 \wedge .. \wedge S_k = \bigcup_{(j_1,..,j_k) \in \prod_i J_i} \{ \bigwedge_{i=1}^{k} \sigma^i_{j_i} \}$$

Par définition si l'un des ensembles S_i est vide alors $S_1 \wedge S_2 \wedge .. \wedge S_n = \emptyset$.

○

Exemple 8.1 --

$$\{\sigma_1,\sigma_2\} \wedge \{\delta_1,\delta_2\} = \{\sigma_1\wedge\delta_1,\sigma_1\wedge\delta_2,\sigma_2\wedge\delta_1,\sigma_2\wedge\delta_2\}$$

∇

Remarque 8.1 -- Soient Σ et Σ' deux ensembles de substitutions telles que la suite des substitutions dans Σ est inconsistante. La conjonction de ces deux ensembles peut ne pas être vide comme le montre l'exemple suivant:

Exemple 8.2 -- Considérons l'exemple précédent avec $\sigma_1=\delta_1=\{x\leftarrow a,y\leftarrow b\}$ et $\sigma_2=\delta_2=\{x\leftarrow b,y\leftarrow a\}$. Bien que $\sigma_1 \wedge \sigma_2$ est inconsistante ($\delta_1 \wedge \delta_2$ de même), nous avons

$$\{\sigma_1,\sigma_2\} \wedge \{\delta_1,\delta_2\} = \{x\leftarrow a,y\leftarrow b\}$$

∇

Nous donnons une propriété de $\wedge$ par rapport à l'union des ensembles.

Lemme 8.1 -- L'opération $\wedge$ sur les ensembles de substitutions est distributive par rapport à $\cup$ ensembliste, nous avons :

$$\Sigma 1 \wedge (\Sigma 2 \cup \Sigma 3) = (\Sigma 1 \wedge \Sigma 2) \cup (\Sigma 1 \cup \Sigma 3)$$

Preuve :

En appliquant les définitions.

□

8.3. Equations et ensembles de solutions.

Nous présentons dans ce paragraphe les équations pour le problème de filtrage. Nous supposons donnée la théorie E dans laquelle nous travaillons.

Définition 8.4 -- Une **équation** pour le problème de filtrage est un couple de termes (M,N) et une théorie E noté $M\ll_E N$. A une équation $M\ll_E N$ est associé un ensemble de solutions, l'ensemble des substitutions σ telles que $\sigma(M) =_E N$.

$$EF(M\ll_E N) = \{\sigma \mid \sigma(M) =_E N\}$$

○

La notation « est choisie par analogie au symbole < arithmétique, dans le sens où on augmente le terme M jusqu'à le rendre égal à N ($\sigma M=N$). Siekmann[Siekmann-78] utilise la convention contraire $N\ll M$ qui se lit "le terme M est plus général que N". Nous préferrons la première notation.

Remarque 8.2 -- L'équation pour le filtrage $x\ll_E f(x)$ admet une solution $\sigma:x\leftarrow f(x)$. Il ne faut pas voir en cette solution un cycle, parceque les variables du terme de droite ne sont pas instanciées: elles peuvent être considérées, à un renommage près, comme distinctes de celles du membre gauche, plus précisement comme des constantes.

Définition 8.5 -- Une équation **normalisée** (ou triviale) est une équation dont le membre gauche est une variable x. A une équation triviale est associe la substitution $\{x\leftarrow t\}$ où t est le membre droit de l'équation.

○

La terminologie "triviale" est justifiée par la remarque précédente (le membre droit d'une équation dans un problème de filtrage est un terme clos), et donc la solution d'une telle équation ($x \ll_E t$) est $x \leftarrow t$.

Notation -- Par la suite, **eq** dénote une équation quelconque $M \ll_E N$, et **eqs** un ensemble d'équations.

Définition 8.6 -- L'ensemble des symboles d'une équations $M \ll_E N$, noté **symb($M \ll_E N$)**, est la réunion des ensembles de symboles de fonction qui apparaissent dans les termes M et N.

○

Si eq est une équation triviale symb(eq) = symb(N) où N est le membre droit de eq.

Nous définissons les systèmes d'équations et leurs solutions.

Définition 8.7 -- Un **système d'équations** est un ensemble d'équations noté $S = M_1 \ll_{E_1} N_1 \wedge \ldots \wedge M_n \ll_{E_n} N_n$. (Nous utilisons le constructeur d'ensemble $\wedge$, qui peut être vu comme un symbole de fonction ACI) L'ensemble des solutions d'un système S (pour le problème de filtrage), est l'ensemble des substitutions, solutions de toutes les équations du système. Cet ensemble de solutions est noté EF(S)

$$EF(S) = \{\sigma, \forall\ i \in [1,n],\ \sigma \in EF(eq_i).$$

Lorsque toutes les équations sont considérées dans la même théorie, ($E_i = E$ pour tout i) l'ensemble des solutions EF(S) peut être noté sans ambiguité $EF_E(M_1 \ll N_1 .. M_k \ll N_k)$ (on ne précise pas la théorie E pour chaque équation mais pour le système tout entier.

○

Nous décrivons maintenant la relation entre l'ensemble de solutions d'un système d'équations et l'ensemble de solutions de chaque équation.

Lemme 8.2 -- L'ensemble des solutions d'un système d'équations est la conjonction des ensembles de solutions de chaques équation du système. Nous avons:

$$EF(eq_1 \wedge eq_2 \wedge .. \wedge eq_n) = EF(eq_1) \wedge EF(eq_2) \wedge .. \wedge EF(eq_n)$$

Preuve :

Montrons l'inclusion $\subseteq$.

Soit σ solution de $EF(eq_1 \wedge eq_2 \wedge .. \wedge eq_n)$, Nous avons donc $\sigma \in EF(eq_i)$ pour tout $i \in [1,n]$, donc $\sigma \wedge .. \wedge \sigma$ (qui est égale à σ) appartient $EF(eq_1) \wedge EF(eq_2) \wedge .. \wedge EF(eq_n)$.

Inversement,

soit $\sigma \in EF(eq_1) \wedge EF(eq_2) \wedge .. \wedge EF(eq_n)$, donc il existe $\{\sigma_i$, $i=1..n\}$ tel que $\sigma = \bigwedge_{i=1}^{n} \sigma_i$, où σ_i est solution de eq_i,

- 1) si $D(\sigma_i) \cap D(\sigma_j) = \emptyset\ \forall\ i \neq j$, alors $\forall$ i, σ est solution de eq_i donc $\sigma \in EF(eq_1 \wedge eq_2 \wedge .. \wedge eq_n)$.
- 2) si $D(\sigma_i) \cap D(\sigma_j) \neq \emptyset$, alors nécessairement, $\forall\ x \in D(\sigma_i) \cap D(\sigma_j)$, on a $\sigma_i(x) = \sigma_j(x)$ (sinon σ n'existerait pas) et donc σ est solution de $EF(eq_1 \wedge eq_2 \wedge .. \wedge eq_n)$.

□

Ce lemme est important car il ramène la recherche des filtres d'un ensemble d'équations, à la recherche des filtres de chaque équation

indépendamment des autres. L'ensemble des solutions de l'ensemble d'équations considérées est la conjonction des ensembles de solutions de chaque équation. Ce résultat implique une strategie de résolution de systèmes d'équations, on verra plus loin d'autres stratégies qui permettent de résoudre un système d'équations sans dissocier les équations, ce qui sera parfois plus judicieux pour certaines applications.

Remarque 8.3 -- Rappelons que $EF(eq_1) \cup EF(eq_2) \neq EF(eq_1 \wedge eq_2)$ comme on le verra dans l'exemple suivant. Nous avons seulement $EF(eq_1) \wedge EF(eq_2) = EF(eq_1 \wedge eq_2)$.

Exemple 8.3 -- Soient $eq_1 = f(x,y) \ll_{\emptyset} f(a,b)$ et $eq_2 = f(x,y) \ll_{\emptyset} f(b,a)$. Nous avons

$$EF(eq_1) = \{x \leftarrow a, y \leftarrow b\}$$
$$EF(eq_2) = \{x \leftarrow b, y \leftarrow a\}$$

et donc

$$EF(eq_1) \cup EF(eq_2) = \{\{x \leftarrow a, y \leftarrow b\}, \{x \leftarrow b, y \leftarrow a\}\}$$

par contre

$$EF(eq_1 \wedge eq_2) = \emptyset$$

car ce système n'a pas de solutions.

∇

Nous définissons maintenant les ensembles de systèmes et leurs ensembles de solutions.

Définition 8.8 -- Une **disjonction de systèmes** S_i pour $i \in I$ est un un ensemble de systèmes notés $S_1 \vee S_2 \vee .. \vee S_i$

L'ensemble des solutions d'une disjonction de systèmes est, par définition, l'ensemble réunion des solutions des systèmes de cette disjonction. Nous avons donc :

$$EF(S_1 \vee S_1 \vee .. \vee S_n) = EF(S_1) \cup EF(S_2) \cup .. \cup EF(S_n)$$

○

Donc en reprenant l'exemple 8.3 $EF(eq_1) \cup EF(eq_2) = EF(eq_1 \vee eq_2)$.

8.4. Transformation de filtricandes

Pour résoudre une équation pour un problème de filtrage, nous allons procéder par transformation d'équations et de systèmes d'équations. Nous ne nous intéressons qu'aux équations et systèmes d'équations, et oublions leurs ensembles de solutions. Le passage entre les équations et leur substitutions sera fait lorsque les équations considérées seront toutes transformées en des équations triviales, ce qui sera alors simple et naturel.

Pour s'assurer de la complétude de la méthode, il est nécessaire que les transformations considérées conservent l'ensemble de solutions. De telles transformations sont appelée des transformations **correctes**.

8.5. Equivalence

Nous appelons **filtricande** une conjonction et/ou une disjonction d'équations. Par exemple $(M_1 \ll_A N_1 \wedge M_2 \ll_B N_2) \vee eqs$ est un filtricande.

Lemme 8.3 -- L'opération $\wedge$ sur les filtricandes est distibutive par rapport à l'opération $\vee$.

$$Q \wedge (S_1 \vee S_2) = (Q \wedge S_1) \vee (Q \wedge S_2)$$

Preuve :

D'après la distributivité de $\wedge$ par rapport à $\cup$.

□

On retrouve dans le lemme précédent la notion d'environnement défini classiquement dans les algorithmes de filtrage et d'unification dans les théories équationnelles et qui correspond à la substitution partielle construite à une étape de l'algorithme.

Définition 8.9 -- Un filtricande F_1 est équivalent à un autre filtricande F_2, et on le note $F_1 \Leftrightarrow F_2$ si et seulement si l'ensemble des solutions $EF(F_1)$ du filtricande F_1 est égale à l'ensemble des solutions $EF(F_2)$ du filtricande F_2.
Si de plus le filtricande F_1 est considéré dans la même théorie E que le filtricande F_2 nous dirons dans ce cas que F_1 est E-équivalent à F_2 et on note $F_1 \Leftrightarrow_E F_2$

○

Exemple 8.4 -- Soit "+" un symbole de fonction commutatif, et soit F le système suivant :

$$F = \{x+y \ll a+b\}$$

C(+) est la théorie ou le symbole "+" est commutatif. Nous avons

$$x+y \ll_{C(+)} a+b \quad \Leftrightarrow \quad x+y \ll_{\emptyset} a+b \vee x+y \ll_{\emptyset} b+a$$

où ø est la théorie vide, c'est-à-dire la théorie où tous les symboles n'ont aucune propriété.

▽

Exemple 8.5 --

$$f(f(x,y),z) \ll_{C(f)} f(a,f(a,b)) \quad \Leftrightarrow$$
$$(f(x,y) \ll_{C(f)} a \ \wedge \ z \ll_{C(f)} f(a,b)) \ \vee$$
$$(f(x,y) \ll_{C(f)} f(a,b) \ \wedge \ z \ll_{C(f)} a)$$

▽

Définition 8.10 -- Nous appelons transformation **correcte**, tout processus qui transforme un filtricande F en un autre filtricande G, tel que $F \Leftrightarrow G$.

○

Chercher l'ensemble des filtres modulo E de M vers N, c'est résoudre l'équation $M \ll N$ dans la théorie équationnelle E c'est-à-dire transfomer cette équation jusqu'à obtenir des équations normalisées (ou triviales). Pour cela nous caractérisons les transformations les plus utilisées sur les filtricande: Décomposition, Fusion et Mutation. La décomposition et la mutation sont des simplifications de filtricandes; la fusion est un arrangement de filtricandes. Cette technique remonte à Martelli et Montanari pour le cas de l'unification dans la théorie vide [Martelli-

Montanari-82] et à C. Kirchner pour le cas des théories équationnelles. Nous leur empruntons les concepts de la décomposition, sauf celui de l'exclusivité qui est original.

Avant de détailler ces trois phases, donnons en l'idée de base:

Décomposition:

$f \in F_{\emptyset}$, l'équation

$$f(t_1, t_2, \ldots, t_k) \ll f(t_1', t_2', \ldots, t_k')$$

a les même solutions que

$$t_1 \ll t_1' \wedge t_2 \ll t_2' \wedge \ldots \wedge t_k \ll t_k'$$

ce que l'on note

$$EF(f(t_1,t_2,..,t_k) \ll g(t_1',t_2',..,t_k')) = EF(t_1 \ll t_1' \wedge t_2 \ll t_2' \wedge \ldots \wedge t_k \ll t_k')$$

D'autres symboles de fonction se comportent de la même manière, de tels symboles sont appelés décomposables.

Nous caractérisons aussi pour cette phase les couples (f,g) de symboles de fonctions pour lesquels l'équation

$$f(t_1,t_2,..,t_k) \ll g(t_1',t_2',..,t_k')$$

n'as pas de solution. De tels couples de symboles sont appellés exclusifs. Dans la théorie vide tout couple (f,g) tq $f \neq g$ est exclusif.

Fusion :

Dans cette transformation, nous regroupons les équations qui ont le même membre gauche afin de vérifier si les deux membres droits sont égaux. Les équations dont le membre gauche est une variable sont collectées pour former la partie triviale du système d'équations. La partie non-triviale est traitée dans l'opération de mutation.

Mutation :

Ce processus est celui de la transformation des équations sur lesquelles on ne peut plus appliquer de décomposition ou fusion et qui forment donc la partie non-triviale du système. Ces transformations sont déterminées par l'ensemble des axiomes que vérifient les symboles de tête des équations considérées, contrairement à la décomposition.

8.5.1. Décompositions

Nous définissons l'ensemble des symboles de fonction qui permettent de simplifier une équation sans se référer aux axiomes de la théorie dans laquelle on résoud cette équation.

Définition 8.11 -- Soit E un ensemble d'axiomes. L'ensemble **Fd(E)** des **symboles E-décomposables** pour un problème de filtrage, est le plus grand sous-ensemble de F telle que :

$$\forall\ f, g \in Fd \quad t = f(t_1,\ldots, t_n), \quad t' = g(t_1', \ldots, t_n')$$
$$f = g \Rightarrow [t = t' \Leftrightarrow \bigwedge_{k=1..n} (t_k = t_k')$$

○

Exemple 8.6 -- Soit F {*, +} où * est distributif gauche par rapport à +:

$$Dg(*,+): x * (y + z) = (x * y) + (x * z)$$

Fd(Dg) = F [Tiden-86].

∇

Exemple 8.7 -- Soit $F = F_{\emptyset} \cup F_c$ ou F_c est l'ensemble des symboles commutatifs. $Fd(C) = F_{\emptyset}$.

∇

Définition 8.12 -- Soit E un ensemble d'axiomes. On appelle ensemble **E-exclusif**, le plus grand sous ensemble Fex(E) des couples (f,g) de F x F tels que si (f,g) apparaissent en tête d'une équation M«N (c'est-à-dire f est le symbole de tête de M et g celui de N), alors {M«N} = ø.

$$Fex[E] = \{(f,g) |\ EF(f(t_1,..,t_n)«g(t_1',..,t_k'))=\emptyset\}$$

avec les notations précédentes.

○

Exemple 8.8 -- Dans les conditions de l'exemple précédent, Fex(C) = {(f,g)∈FxF, f≠g}.

∇

Nous décrivons la décomposition par des règles de réécriture, où « est un nouveau symbole de fonction. Nous rappelons que ø est la substitution vide et que ø∧eq=ø et ø∨eq=eq.

Soit fd ∈ Fd et (fx,gx) ∈ Fex

D1: $fd(t_1,..,t_n)«fd(t_1',..,t_k') \rightarrow t_1«t_1' \wedge ... \wedge t_k«t_k'$

D2: $fx(t_1,..,t_n)«gx(t_1',..,t_k') \rightarrow \emptyset$

La décomposition d'une équation se généralise à la décomposition d'un filtricande en décomposant toutes ses équations.

La décomposition est une transformation correcte comme le montre le lemme suivant.

Lemme 8.4 -- $\rightarrow_{D1 \cup D2}$ est une transformation correcte.

Preuve :

D'après la définition de Fd et Fex.

□

8.5.2. Fusion

Dans cette phase nous regroupons les équations triviales ($x«_E t$ et $y«_E t'$) pour former ce qu'on appelle la partie triviale du système.

Définition 8.13 -- On appelle **partie triviale** d'un filtricande l'ensemble des équations triviales de ce filtricande. La **partie non-triviale** est le reste du filtricande.

○

Au cours de la transformation de fusion, s'il existe dans le filtricande considéré deux équations triviales $x \ll_E t_1$ et $x \ll_E t_2$, une seule équation est transférée dans la partie triviale ($x=t_1$ par exemple), et l'équation $t_1 \ll_E t_2$ est alors ajoutée dans la partie non-triviale du système. Nous avons:

Définition 8.14 -- La règle de réécriture pour la fusion est la suivante:

$$F1: \quad x \ll_E t_1 \wedge x \ll_E t_2 \rightarrow x \ll_E t_1 \wedge t_1 \ll_E t_2$$

○

Remarque 8.4 -- t_1 et t_2 sont deux termes clos donc l'équation $t_1 \ll_E t_2$ peut être résolue par un algorithme spécial de test de E-égalité. L'équation $t_1 \ll_E t_2$ ne peut avoir que deux solutions ø ou Identité, dans le premier cas la fusion termine en échec, et dans le second $x \ll_E t_1 \wedge x \ll_E t_2$ est équivalent à $x \ll_E t_1$. Cela peut être axiomatisé par les règles de réécriture suivantes:

$$F2: \quad x \ll_E t_1 \wedge x \ll_E t_2 \rightarrow x \ll_E t_1 \quad \text{si} \quad t_1 =_E t_2$$

ou bien

$$F'2: \quad x \ll_E t_1 \wedge x \ll_E t_2 \rightarrow x \ll_E t_1 \wedge x \ll_E t_2 \wedge t_1 =_E t_2$$

$$F3: \quad eq \wedge ø \rightarrow ø$$

$$F4: \quad eq \wedge Id \rightarrow eq$$

$$F5: \quad eq \wedge eq \rightarrow eq$$

Cette transformation, comme la décomposition conserve l'équivalence des filtricandes.

Lemme 8.5 -- $\rightarrow_{F1}$ et $\rightarrow_{F2UF3UF4}$ sont des transformation correctes.

8.5.3. Mutation

La mutation est la transformation de la partie non-triviale. Elle dépend de l'ensemble des axiomes E considérés. Pour un filtricande U, Mut(U) est l'opération qui à U fait correspondre un autre filtricande U' tel qu'on sait résoudre les équations de U'. Nous exigons de la transformation de mutation d'être une transformation correcte pour assurer l'égalité des ensembles de solutions des filtricandes avant et après la mutation. La mutation n'est pas unique en général.

Définition 8.15 -- La mutation mut est la transformation des filtricandes F composés d'équations de la forme $f(..) \ll g(..)$ avec f, g des symboles non décomposable et (f,g) non exclusif, et tel que EF(F) = EF(mut(F)).

○

Pour un ensemble d'axiomes E, les techniques de caractérisation de E-Mut diffèrent d'un ensemble d'axiomes à un autre et reposent souvent sur des principes fort différents les uns des autres. Ils ont tous le même but: simplifier le filtricande. C. Kirchner [Kirchner85] a décrit plusieurs méthodes de mutation d'un ensemble d'équations dans le cas de l'unification, ces méthodes peuvent aussi être utilisées pour le filtrage.

Nous donnons dans le chapitre "Union de théorie", une méthode originale de mutation basée sur la transformation des théories. Une méthode générale de mutation est la surréduction [Hullot-80a,Jouannaud,etal.-83,Kirchner-85,Rety,etal.-85,Rety-87].

Dans le cas du filtrage, il suffit de connaitre la mutation d'une équation M«N où les sous-termes de M sont tous des variables.

Lemme 8.6 -- Soient M et N deux termes. $M=f(t_1,..t_n)$ et $N=g(t_1',..,t_k')$, et soient $x_1,..,x_n$ des variables qui n'appartiennent pas à V(M), nous avons

$$f(t_1,..t_n)«N \Leftrightarrow f(x_1,..,x_n)«N \wedge t_1 \leftarrow x_1 \wedge .. \wedge t_n \leftarrow x_n$$

ce filtricande restreint à V(M).

avec la règle suivante

$$F6:\ x«D \wedge G \leftarrow x \rightarrow G«D$$

Nous ne détaillons pas la sémantique de ↩. G↩x peut être considérée comme une équation particulière.

Donnons un exemple avant de donner la preuve.

Exemple 8.9 -- Pour résoudre l'équation $f(a,v)«_{C(f)}f(a,b)$ il suffit de résoudre le système suivant

$$f(x,y)«_{C(f)}f(a,b) \wedge a \leftarrow x \wedge v \leftarrow y \Leftrightarrow$$
$$((x«_{C(f)}a \wedge y«_{C(f)}b) \vee (x«_{C(f)}b \wedge y«_{C(f)}a)) \wedge$$
$$a \leftarrow x \wedge v \leftarrow x \Leftrightarrow$$
$$(x«_{C(f)}a \wedge y«_{C(f)}b \wedge a \leftarrow x \wedge v \leftarrow y) \vee$$
$$(x«_{C(f)}b \wedge y«_{C(f)}a \wedge a \leftarrow x \wedge v \leftarrow y) \Leftrightarrow$$

en appliquant la règle F6

$$(a«_{C(f)}a \wedge v«_{C(f)}b) \vee (a«_{C(f)}b \wedge v«_{C(f)}a) \Leftrightarrow$$
$$(Id \wedge v«_{C(f)}b) \vee (\emptyset \wedge v«_{C(f)}a) \Leftrightarrow$$
$$(v«_{C(f)}b \vee \emptyset \Leftrightarrow$$
$$(v«_{C(f)}b)$$

Les variables x_i des équations $t_i \leftarrow x_i$ ne sont pas instantiés, ce sont des variables intermédiaires qui disparaissent une fois l'équation $f(x_1,..,x_n)«N$ est complètement transformée. En effet pour chaque équation du type $t_i \leftarrow x_i$, il existe une équation $x_i«d$, après transformation de $f(x_1,..,x_n)«N$, ce qui donne, en appliquant la règle F6 $t_i«d$.

∇

Donnons maintenant la preuve du lemme.

Preuve :

Soit $M'=M[t_j \leftarrow x_j]$

1) Montrons $EF(M[t_j \leftarrow x_j]«N \wedge t_i«x_1,\wedge .. \wedge t_n«x_n) \subseteq EF(M«N)$. Soit σ dans premier ensemble,

$$\sigma=\{t_1«x_1,..,t_1«x_1\} \cup \sigma_1,$$

avec $\sigma_1(M')=N$, en appelant σ_2 l'ensemble $\{t_1«x_1,..,t_1«x_1\}$, nous avons $\sigma=\sigma_2\sigma_1$, et $\sigma(M) = \sigma_1(M') = N$.

2) Inversement,

Soit σ une substitution telle que $\sigma M=N$. on veut montrer que $\sigma \in EF(M'\ll N \wedge t_1\ll x_1 \wedge .. \wedge t_n\ll x_n)$ qui est égale à $\{M'\ll N\}\cup\sigma_2$, or en appliquant F6 sur ce système on trouve $M\ll N$.

□

Remarque 8.5 -- On peut voir le lemme 8.6 comme l'application de la règle suivante

$$f(x_1,..,x_n)\ll N \rightarrow D(x_1,..,x_n,N)$$

sur une équation du type

$$f(t_1,..,t_n)\ll N$$

ce qui donne $D(t_1,..,t_n,N)$, c'est cette application de la règle qu'on note par

$$(f(x_1,..,x_n)\ll N) \wedge (t_1\ll x_1 \wedge..\wedge t_n\ll x_n)$$

Dans le cas de l'exemple précédent, nous avons la règle

$$f(x,y)\ll f(a,b) \rightarrow (x\ll a \wedge y\ll b) \vee (x\ll b \wedge y\ll a)$$

que l'on applique à $f(a,v)\ll f(a,b)$, on obtient

$$(a\ll a \wedge v\ll b) \vee (a\ll b \wedge v\ll a)$$

ce que nous exprimons dans le lemme 8.6 par

$$[(x\ll a \wedge y\ll b) \vee (x\ll b) \wedge (y\ll a)] \wedge (a\ll x) \wedge (v\ll y)$$

en utilisant la règle F6, nous obtenons

$$(a\ll a \wedge v\ll b) \vee (a\ll b \wedge v\ll a)$$

8.5.4. Algorithme de Filtrage

Un algorirhme de filtrage est donné par le système de réécriture composé des règles de réécritures définies dans les paragraphes précédents et qui sont une formalisation des transformations décrites dans le paragraphe précédent.

F1) Fusion

F2: $x\ll_E t_1 \wedge x\ll_E t_2 \wedge t_1\ll_E t_2 \rightarrow x\ll_E t_1$

F3: $eq \wedge \emptyset \rightarrow \emptyset$

F4: $eq \wedge Id \rightarrow eq$

M_1) Mutation

M1: $F \rightarrow mut(F)$

F3) Décomposition

La décomposition est un cas particulier de règle M1), c'est la mutation associé à la théorie vide. Pour les symboles $fd \in Fd$ et (fx,gx) dans Fex, elle peut être définie comme suit:

D1: $fd(t_1,..,t_k) \ll fd(t_1',..,t_k') \rightarrow t_1 \ll t_1' \wedge ... \wedge t_k \ll t_k'$

D2: $fx(t_1,..,t_n) \ll gx(t_1',..,t_k') \rightarrow \emptyset$

La règle M1 est en réalité un ensemble de règles décrivants la mutation d'un unificande pour la théorie considérée. Le lemme suivant nous assure la correction de la méthode.

Lemme 8.7 -- Soit U un filtricande et U' le filtricande obtenu après l'application de l'une des transformations précédentes, alors $U \Leftrightarrow U'$.

Preuve :

Toutes les transformations utilisées sont correctes, comme nous l'avons montrées dans le paragraphe précédent.

□

nous avons la complétude de la méthode lorsque la mutation est décrite complètement.

Nous regroupons toutes les règles de réécriture décrivant la décomposition, fusion et la mutation. Le système de réécriture obtenu est complété en un système convergent. Dans le cas d'existence de tel système, nous avons un algorithme de filtrage dans la théorie considérée. La terminaison de cet algorithme de filtrage et sa correction découlent de la convergence du système. Donnons un exemple pour le cas de la théorie vide, d'autres exemples de théories plus complexes peuvent être trouvés dans le paragraphe 8.7 (Exemples de filtrage par réécriture).

Exemple 8.10 -- Soit F l'ensemble des symbole de fonction {f,g,a,b}, f

et g sont d'arité deux, a et b des constante. Considérant le système de réécriture qui axiomatise les trois phase de l'algorithme de filtrage où "echec" est la substitution vide, Id est l'identité.

V1: a « b == échec
V2: (x « b) ∧ (x « a) == échec
V3: f(x, y) « g(z, u) == échec

V4: f(x, y) « f(z, u) == (x « z) ∧ (y « u)
V5: g(x, y) « g(z, u) == (x « z) ∧ (y « u)
V6: a«a == Id
V7: b«b == Id

V8: échec ∧ x == échec
V9: x ∧ Id == x
V10: x ∧ x == x

Les trois premières règles (V1 V2 V3) décrivent l'exclusivité, les quatre suivante (V4 V5 V6 V7) la décomposition et les deux dernière (v8 V7) decrivent les propriété de l'opération ∧. La complétion de ce système avec ∧ symbole AC engendre le système suivant:

V1: a « b → échec
V2: (x « b) ∧ (x « a) → échec
V3: f(x, y) « g(z, u) → échec
V11: b « a → échec

V4: f(x, y) « f(z, u) → (x « z) ∧ (y « u)
V5: g(x, y) « g(z, u) → (x « z) ∧ (y « u)
V6: a«a → Id
V7: b«b → Id

V8: échec ∧ x → échec
V9: x ∧ Id → x
V10: x ∧ x → x

Ce système convergent peut être utilisé comme un algorithme de filtrage dans la théorie considéré, il suffit de normaliser le terme M«N pour trouver le filtre de M vers N. donnons une sortie de la normalisation utilisant le système REVE:

Résolvons l'équation f(x,g(a,x))«f(a,g(a,b))

Please enter the term for which you would like the normal form computed,
terminated by <ESC>:
f(x,g(a,x))«f(a,g(a,b))

The sequence of term réductions leading to the normal form of your term is:
f(x, g(a, x)) « f(a, g(a, b))
(g(a, x) « g(a, b)) ∧ (x « a)
(a « a) ∧ (x « a) ∧ (x « b)
(a « a) ∧ échec
échec

Considérons l'équation suivante

$$f(f(x,y),f(g(a,b),f(x,y)))«f(f(a,b),f(g(a,b),f(a,b)))$$

Please enter the term for which you would like the normal form computed,
terminated by <ESC>:
f(f(x,y),f(g(a,b),f(x,y)))«f(f(a,b),f(g(a,b),f(a,b)))

The sequence of term réductions leading to the normal form of your term is:
f(f(x, y), f(g(a, b), f(x, y))) « f(f(a, b), f(g(a, b), f(a, b)))
(f(g(a, b), f(x, y)) « f(g(a, b), f(a, b))) ∧ (f(x, y) « f(a, b))
(f(x, y) « f(a, b)) ∧ (f(x, y) « f(a, b)) ∧ (g(a, b) « g(a, b))
(f(x, y) « f(a, b)) ∧ (g(a, b) « g(a, b))
(g(a, b) « g(a, b)) ∧ (x « a) ∧ (y « b)
(a « a) ∧ (b « b) ∧ (x « a) ∧ (y « b)
(b « b) ∧ (x « a) ∧ (y « b) ∧ Id
(b « b) ∧ (x « a) ∧ (y « b)
(x « a) ∧ (y « b) ∧ Id
(x « a) ∧ (y « b)

Considérant deux termes égaux:

Please enter the term for which you would like the normal form computed,
terminated by <ESC>:
f(a,b)«f(a,b)

The sequence of term réductions leading to the normal form of your term is:
f(a, b) « f(a, b)
(a « a) ∧ (b « b)
(b « b) ∧ Id
b « b
Id

∇

Prouver la terminaison de l'algorithme de filtrage revient à prouver la terminaison du système de réécriture considéré, cela peut se faire en utilisant les ordres RDO, RPO... Remarquons que si la condition nécesssaire donnée par C. Kirchner pour la terminaison est vérifiée, le système de réécriture considéré termine. Cette condition s'énonce ainsi:

Définition 8.16 --[Kirchner-85]-- Soit μ une mesure de complexité des filtricandes (une application de la classe des filtricandes dans N) relative à la théorie E, Une mutation mut se termine si et seulement si

$$\mu(Mut(F)) < \mu(F)$$

○

Lemme 8.8 -- Soit mut une transformation qui se termine, alors la forme normale d'un filtricande F est composée uniquement d'équations triviales. A cet ensemble d'équations est associé un ensemble de substitutions égal à l'ensemble des solutions de F.

Preuve :

Ce résultat est une conséquence des systèmes de réécriture à terminaison fini.

□

8.6. Exemples de Filtrage

Nous étudions dans les paragraphes suivants quelques cas de filtrage utiles en réécriture et en programmation logique. Pour chaque cas nous appliquons l'algorithme standard de filtrage, pour cela nous décrivons les ensembles Fex(E) et Fd(E) et la phase de mutation.

Notons que pour le cas des théories EUI (contenant l'axiome d'idempotence) il suffit de donner la mutation pour les équations dont les termes ont le même symbole de tête et ceci d'après le résultat suivant:

Lemme 8.9 --[Mzali-84]-- Soit M tel que top(M) vérifie l'axiome d'idempotence. $M \ll_{EUI} N \Leftrightarrow M \ll_{EUI} top(M)(N,N)$.

La phase de mutation est complètement développée pour ces différents exemples. Les termes sont considérés avec leurs formes normmales FN définie dans le chapitre précédent.

Lorsque nous parlons de théorie E, on suppose que l'ensemble des symboles de fonction F est l'ensemble Fe: tous les symboles de fonctions vérifient l'axiome E.

8.6.1. Théorie vide

L'ensemble de symboles de fonctions est noté Fv

Fd = F

Fex ={(f,g) / f ≠ g}

Cette théorie est la plus simple à cause de l'absence de la phase de mutation. Tous les symboles de fonctions sont décomposables et l'algorithme de filtrage n'a que deux phases qui se succèdent: décomposition puis fusion. Nous verrons dans le paragraphe 8.5.4 une impalntation de ce cas par les règles de réécriture.

8.6.2. Théorie associative

L'ensemble des symboles de fonction associatif est noté

Fa

Fd = ø

Fex = {(f,g) / f ≠ g}

La mutation est appliqué aux équations de symbole de tête appartenant à {(f,g) / f = g}. Elle correspond à la résolution des équations dans l'ensemble des structures de listes. Cette résolution peut être décrite par un simple parcours de liste, par exemple /xy/=/abc/ admet deux solutions {x←a,y←bc} et {x←ab,y←c}.

8.6.3. Théorie commutative

L'ensemble des symboles de fonction commutatif est noté

Fc

Fd = ø

Fex = {(f,g) / f ≠ g}

La mutation est appliquée aux équations dont le symbole de tête est dans {(f,g) / f = g}, et c'est la résolution des équations dans les couples

permutatifs. [x,y]=[a,b] admet deux solutions {x←a,y←b} et {x←b,y←a}.

8.6.4. Théorie Idempotente

L'ensemble des symboles de fonction idempotent est noté

Fi

Fd = ø

Fex = ø

Dans ce cas la mutation est appliquée à l'ensemble tout entier, elle correspond à la résolution des équations dans les structures de couple permutatif sans répétition. la mutation de l'équation $f(x,y)\ll_I g(a,b)$ est $(x\ll_I y \wedge y\ll_I b)\vee(x\ll_I g(a,b)\wedge y\ll_I g(a,b))$.

8.6.5. Théorie associative-commutative

L'ensemble des symboles de fonction associatif-commutatif est noté Fac

Fd = ø

Fex = {(f,g) / f ≠ g}

La AC-forme normale d'un terme utilise la structure des multiensembles, comme cela a été presenté au chapitre 7. $f\{\{t_1,\ldots t_k\}\}$ est la forme normale de tous les termes AC-égaux au terme $f(t_1,f(t_2,\ldots t_k))$, les ti n'ont pas de symboles de tête égaux à f. la mutation est appliquée aux équations de symbole de tête dans {(f,g) /f = g}, elle correspond à la résolution des équations dans la structure de multiensemble qui est la résolution d'équations diophantiennes à coefficients dans N (appelée partition de mots abéliens) [Hullot-80].

En considérant deux termes M et N sous leurs formes normales AC définies dans le chapitre précédent

M = f{{x,y}}
N = f{{c ,c ,b}}

résoudre l'équation

$$M\ll_{AC} N$$

revient à résoudre

$$(b^p c^{p'})(b^q c^{q'}) = b^1 c^2$$

Ce qui correspond à l'équation diophantienne suivante

$$p + q = 1$$

$$p' + q' = 2$$

Les seules solutions possibles (p et p' ne pouvant pas être nuls simultanément, ils peuvent être représentés comme suit [Mzalil983]:

	x	y	x	y	x	y	x	y	x	y
c	2=p'+ q'		2	0	0	2	1	1	1	1
b	1=p + q		0	1	1	0	0	1	1	0

solution: x = $f\{\{c^{p'},b^p\}\}$

y = $f\{\{c^{q'},b^q\}\}$

↓

x=c

y=f{{c ,b}}

8.6.6. Théorie associative-idempotente

L'ensemble des symboles de fonction Associatif-idempotent est noté Fai

Fd = ø

Fex = ø

Fmut = Fai

La mutation dans ce cas est la résolution des équations sur les listes sans répétitions immédiates. L'équation //xy//=//abc// peut être résolue comme pour le cas A avec en plus la simplification //xx//=//x//.

8.6.7. Théorie Commutative-Idempotente

L'ensemble des symboles de fonction CI est noté Fci

Fd = ø

Fex = ø

Fmut = Fci

La mutation dans ce cas est la résolution d'équations dans la structure des couples permutatifs sans répétition. L'équation [[x,y]]=[[a,b]] admet trois solutions

x=a, y=b
x=b, y=a
x=y=[[a,b]]

8.6.8. Théorie Associative-Commutative-Idempotente

L'ensemble des symboles de fonction ACI est noté Faci

Fd = ø

Fex = ø

La mutation dans le cas ACI correspont à la résolution d'équation booléenne. Les équations en p et q du cas AC sont considérées dans l'ensemble $\mathbb{B}$, c'est-à-dire que p et q $\in \{0,1\}$. L'exemple du cas AC devient dans le cas ACI

M = f{x,y}
N = f{c ,b} (puisque {c ,c ,b} = {c ,b]

résoudre l'équation

$$M \ll_{ACI} N$$

revient à trouver toutes les matrices 2où aucune ligne ni colonne n'est nulle, et telle que dans chaque ligne et chaque colone aumoins une valeur est non nulle [Mzali-84,Mzali-85].

	x	y	x	y	x	y	x	y	x	y	etc..
c	p	q	1	1	1	0	1	1	1	1	
b	p'	q'	1	1	1	1	0	1	1	0	

solution: $x = f\{c^{p'}, b^{p}\}$

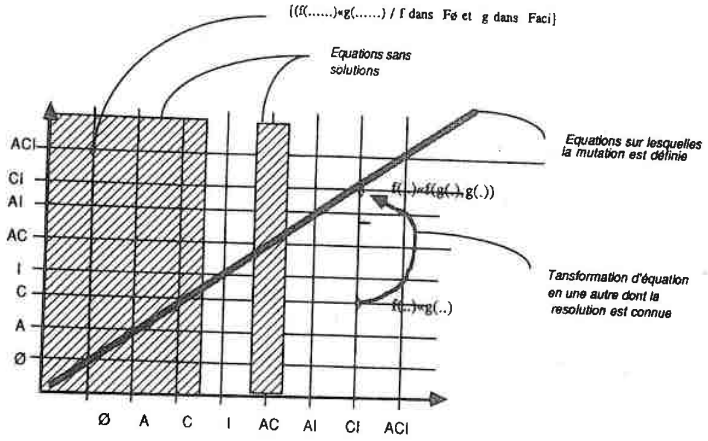
$y = f\{c^{q'}, b^{q}\}$

↓

x=c

y=f{c ,b}

8.6.9. Tableau pour A, C ou I



8.6.10. Théorie distributive gauche (ou droite)

Dans ce cas il n'existe pas de structure évidente pour caractériser les formes normales modulo la distributivité gauche, mais la mutation a pu être caractérisée dans tous les cas possibles [Arnborg-Tiden-85]. Nous présontons dans le paragraphe suivant une étude du problème du filtrage distributif gauche (ou droit) qui fournit un algorithme de filtrage utilisant les règles de transformation de filticandes décrites précédemment.

Matching with distributivity *

Jalel Mzali

Centre de Recherche en Informatique de Nancy
Campus Scientifique B.P. 239
54506 Vandoeuvre Les Nancy Cedex
and
Greco de Programmation
(C.N.R.S)

ABSTRACT

We study matching problems for one-sided distributivity. A general method is presented to build up a matching algorithm in an equational theory, and illustrated by the case of one-sided distributivity. Using this algorithm, (right and left) distributivity matching is shown decidable and a method to compute distributive matches is given.
Key words: matching, distributivity, equational theory, decomposition, merging, mutation.

1. Introduction

Solving an equation $t_1 == t_2$ modulo a set of axioms E (or unifying t_1 and t_2 in the equational theory E) consists of finding a substitution σ such that $\sigma(t_1) =_E \sigma(t_2)$.

In many cases, such as program transformation [Der84] or Knuth-Bendix completion [JoK84], we are interested in a matching problem, i.e. a substitution σ is needed such that $\sigma(t_1) = t_2$. Matching is also the basic mechanism of languages based on rewriting such as OBJ2 [FGJ85]. Of course if a unification algorithm is known, it can be applied for computing matches. However, efficiency is crucial in applications such as Knuth-Bendix completion or rewriting-based languages. Therefore, it is still advantageous to find matching algorithms.

The axiom of distributivity (left and/or right) is especially important in theorem proving applications [KaN85] [Hsi85] and in other kind of applications as well [BeK84]. Decidability of unification under distributivity (left and right) is open [Sza82], but one sided-distributivity (left or right) has been shown decidable by Arborg & Tiden who gave a complete algorithm [ArT85]. On the other hand matching under distributivity has not yet been studied. Starting from a new method of constructing matching algorithms, we develop an efficient algorithm for the left (or right) distributivity. Furthermore, we give a simple method to decide the equality problem modulo left (or right) distributivity. Finally we solve the open problem of matching under left and right distributivity. However, our algorithm is derived from simple general consideration upon finite congruence classes, which leads to a somewhat inefficient algorithm.

2. Substitutions and Matches

We assume the reader to be familiar with the basic notions of term rewriting systems [HuO80], but recall some definitions related to our problem.

Definition 2.1 — Let F be a graded set of **function symbols**, and X a set of **variables**. M(F,X) is the free algebra over X. Elements of M(F,X) are called **terms**.

Definition 2.2 — **Substitutions** σ of M(F,X) are defined to be endomorphisms of M(F,X) equal to the

In Proc. 8th Conf. On Automated Deduction. Oxford LNCS 230

identity on X except on a finite subset of it, called the **domain** $D(\sigma)$ of the substitution. $I(\sigma)$ denotes the set of variables occurring in the terms $\sigma(x)$ for all variables x in the domain of σ. $\sigma|W$ is the restriction of σ on the subset W of $D(\sigma)$. Composition of substitutions σ and ρ is written $\rho\sigma$. We say that ρ factors σ, if there is ξ such that $\sigma = \rho\xi$.

Definition 2.3 — We call an **axiom** any pair of terms $\{t_1,t_2\}$ written $t_1 = t_2$. For a finite set of axioms E, the **E-equality** is the smallest congruence on M(F,X) containing all pairs $\{\sigma(t_1),\sigma(t_2)\}$ where $t_1 = t_2$ is any axiom of E and σ any substitution. The E-equality is written $=_E$.

Definition 2.4 — We denotes by $\lesssim_E$ the quasi-ordering on M(F,X) defined by $t_1 \lesssim_E t_2$ iff $t_2 =_E \sigma(t_1)$, where σ called an **E-match**, is a substitution from t_1 to t_2. Let $V \subseteq X$, we say that a substitution σ factors σ' modulo E on V, written as $\sigma \lesssim_E \sigma'$ [V], iff there exists a substitution σ'' such that $\forall x \in V$, $\sigma'(x) =_E \sigma''(\sigma(x))$.
A substitution σ is written $\{(z_1 \leftarrow t_1), \ldots (z_n \leftarrow t_n)\}$. When $E = \emptyset$, $\lesssim_E$ is written as $\lesssim$.

Unlike the standard case of the empty theory, there may be (infinitely) many independent matches from t_1 to t_2 in general. We are therefore interested in finding a basis of the set of matches whenever one exists [FaH83]. We now give the definition of a complete set of E-matches.

Definition 2.5 — [FaH83]— Let M and N be two terms, V(M) the set of variables of M. The set of the E-matches from M to N is written $\{M{=}{=}N\}_E$. Let W be a set of variables containing V(M). A **complete set** of E-matches from M to N outside W is a set of substitutions denoted $C\{M{=}{=}N\}_E$ such that

- $\forall \sigma \in C\{M{=}{=}N\}_E \quad D(\sigma) \subseteq V(M)$ and $I(\sigma) \cap (W\text{-}V(N)) = \emptyset$ (Protection of variables)
- $C\{M{=}{=}N\}_E \subseteq \{M{=}{=}N\}_E$ (correctness)
- $\forall \sigma \in \{M{=}{=}N\}_E$, there exists $\rho \in C\{M{=}{=}N\}_E$ so that $\rho \lesssim_E \sigma$ [V] (completeness)

Moreover $C\{M{=}{=}N\}_E$ is minimal if

- $\forall \sigma, \sigma' \in C\{M{=}{=}N\}_E \quad \sigma \lesssim_E \sigma' \Rightarrow \sigma = \sigma'$ (minimality)

3. Transformation of Equations

We now introduce equations and systems of equations that constitute the basis of our framework.

Definition 3.1 — We call an **equation** any pair of terms written M==N. A **trivial equation** is an equation whose first term is a variable. An **E-solution** (or E-match) of an equation M==N is any E-match from M to N.
A **system** is a set of equations $U = \{M_i{=}{=}N_i \mid i \in I\}$. An E-solution of U is any substitution σ such that σ is an E-solution of each equation $M_i{=}{=}N_i$. We denote by $\{M_i{=}{=}N_i \mid i \in I\}_E$ the set of all solutions.

Definition 3.2 — An **environment** is a system of trivial equations $\{z_i{=}{=}t_i \mid i \in I\}$ where z_i appears at most once, as the left part of an equation. The Domain of an environment e, written D(e) is the set of variables having an occurrence as left part of an equation.
Given an environment $e = \{z_i{=}{=}t_i \quad i \in I\}$, we define an associated substitution σ_e of domain D(e) such that

$\sigma_e(z_i) =$ if t_i is a variable and $t_i \notin D(e)$ then t_i
else $\sigma_e(t_i)$

Remark 3.1 — σ_e is well defined, if there is no cycle in the relation $(x, \sigma_e(x))$ This is the case in our study because:

1) each added variables is a new variable, one which, does not occur anywhere else in the set of equations,
2) the right hand side of an equation can be considered as a ground term. As a consequence, environments like $\{(x{=}{=}x)\}$ or $\{(x{=}{=}y),(y{=}{=}x)\}$ are not possible.

In some cases like distributivity matching, we need to introduce new variables during the matching process. This provides environments such as $(x{=}{=}y,y{=}{=}z)$ where y is the new variable. This is why we need a

recursive definition of σ_e as given above.

Example 3.1 — for $e = \{(x{=}{=}a), (y{=}{=}v), (v{=}{=}f(a,b))\}$,

$\sigma_e(x) = a$,
$\sigma_e(z) = z$,
$\sigma_e(y) = f(a,b)$ and
$\sigma_e(a) = a$.

Our matching method amounts to transforming any system of equations until it contains only trivial equations. Correctness and completeness of our transformation are consequences of using an equivalence relation among sets of equations.

Definition 3.3 — We say that systems $S = \{M_i{=}{=}N_i \mid i \in I\}$ and $S' = \{M_j'{=}{=}N_j' \mid j \in J\}$ are E-**equivalent** for the matching problem, and we write $S \Leftrightarrow_E S'$, iff $\{M_i{=}{=}N_i \mid i \in I\}_E = \{M_j'{=}{=}N_j' \mid j \in J\}_E$.

A system can contain trivial equations, and in that case the system is divided into two parts called **trivial part** (containing only trivial equations), and **non-trivial part** (containing the others).

The base of our method is to increase the trivial part and decrease the non-trivial one while respecting the equivalence of equations. This is done by looking at head symbols in the equations, as explained now.

Some symbols of F may behave, with respect to simplification of equations, as if E is an empty theory. Such symbols are called decomposable [Kir85]:

Definition 3.4 — Given a set of axioms, the set Fd(E) of **E-decomposable** symbols for the matching problem, is the largest subset of F such that

$\forall f, g \in Fd \quad t = f(t_1,\ldots, t_k), \quad t' = g(t_1',\ldots, t_k')$

$$f = g \Rightarrow \{(t = t') \Leftrightarrow_E \{(t_i = t_i')_{i=1..k}\}\}.$$

Similarly, some symbols of F may behave, for the problem of non existence of solutions, as if E is non empty:

Definition 3.5 — Given a set of axioms, the **E-exclusive** set Fex(E) is the largest subset of F×F such that for two terms $f(t_1,..,t_k)$ and $g(t_1',..,t_{k'}')$, $(f,g) \in$ Fex:

$$f \neq g \Rightarrow \{f(t_1,..,t_k){=}{=}g(t_1',..,t_{k'}')\}_E = \emptyset.$$

4. A General Scheme for Matching Modulo an Arbitrary Set of Equations E.

Computing a set of matches of two terms M and N in the equational theory defined by E, is done by transformation of the set of equations $EQ_1 = \{M_i{=}{=}N_i \mid i \in I\}$ into a set of equations EQ_n E-equivalent to EQ_1, and which contains only trivial equations. This transformation consists of applying repeatedly the three following steps: merging, decomposition and mutation. In what follows we briefly present these three steps which constitute the main phases of our matching algorithm in an equational theory E.

A more detailed presentation of the method in the general case of unification is given by C. Kirchner [Kir85]. A study of the same method for associative-commutative and/or idempotent matching can be found in [Mza85].

Merging step:

During this step we group together trivial equations, to form the trivial part of the system. If two trivial equations have the same left part like $(x{=}{=}t_1)$ and $(x{=}{=}t_2)$, then, $(x{=}{=}t_1)$ stays in the trivial

part while $t_1 == t_2$ goes to the non-trivial part.

$$\begin{cases} (x+y)==(a+b) \\ x==a \\ y==b \\ z==t_1 \end{cases} \rightarrow \begin{cases} (x+y)==(a+b) \\ a==t_1 \\ \hline x \leftarrow a \\ y \leftarrow b \end{cases}$$

Decomposition step:

Non-trivial equations that have the same decomposable top symbol are transformed into a systems of simpler equations. During this step, only the decomposable and/or exclusive symbols are used, which insures the equivalence of the starting system and of the decomposed system. If for example f belongs to Fd and + does not, the system:

$$\{ (f(x,y)==f(a,b)), ((x+y)==(a+b)) \}$$

will be decomposed into the E-equivalent system

$$\{x==a,\ y==b,\ (x+y)==(a+b)\}$$

On the other hand, if we have an equation $f(t_1,..,t_k)==g(t_1',..,t_k')$ with $(f,g) \in$ Fex, we conclude that the starting system has no solution.

Mutation step:

The non-trivial part of the system only deals with the last two steps. The mutation step consists of transforming equations that cannot be transformed by the previous steps because their top symbols are not decomposable or do not belong to Fex. These transformations are specific to the theory E, unlike the two previous steps which are purely syntactic. Of course, they must transform a set of equations into an E-equivalent one.

For example, let us start from the system obtained at the previous step be $\{(x+y)==(a+b),\ x \leftarrow a,\ y \leftarrow b\}$, and let us assume that the symbol "+" is commutative. The mutation of this system yields a disjunction of two systems:

$$\begin{cases} \dfrac{(x+y)==(a+b)}{\begin{matrix} x \leftarrow a \\ y \leftarrow b \end{matrix}} \rightarrow \begin{cases} x==a \\ y==b \\ \hline x \leftarrow a \\ y \leftarrow b \end{cases} \\ \qquad \text{or} \\ \qquad \begin{cases} x==b \\ y==a \\ \hline x \leftarrow a \\ y \leftarrow b \end{cases} \end{cases}$$

Our matching algorithm modulo E is a repetitive application of the three previous steps, as long as the non-trivial part of the system is not empty. A sufficient condition for this algorithm to *stop* is that the mutation step introduces equations of a size strictly smaller than the size of the previous equations. The *correctness* of the algorithm follows from the proof that each transformation step maintains the equivalence of the systems.

5. Case of Left Distributivity

$$\mathbf{Dl}:\ (x * y) + (x * z) = x * (y + z)$$

We only consider terms which are canonical with respect to the canonical system $\mathbf{DL} = \{(x*y)+(x*z) \rightarrow x*(y+z)\}$. Non-canonical terms are reduced in their canonical form before the mutation process starts.

5.1. Merging

As described above, this step is completely independent from the Axiom Dl.

5.2. Decomposition

We first characterize Fd and Fex:

Proposition 5.1 — The symbols + and * are decomposable, thus

$$\{(t_1+t_2==t_3+t_4)\} \Leftrightarrow_{Dl} \{(t_1==t_3),\ (t_2==t_4)\}$$
$$\{(t_1*t_2==t_3*t_4)\} \Leftrightarrow_{Dl} \{(t_1==t_3),\ (t_2==t_4)\}$$

Proof :

See Arnborg and Tiden [ArT85]. []

Proposition 5.2 — Let Fr be the subset of F which contains all symbols not involved in the distributive axiom, then

$$\text{Fex} = (\text{Fr} \times \text{F}) \cup (\text{F} \times \text{Fr})$$

Proof :

It is easy to see that (Fr x F) ∪ (F x Fr) ⊆ Fex. The reversed inclusion follows from the fact that equations of the form +(..)==*(..) or *(..)==+(..) need a mutation process as detailed in subsection 5.3. []

5.3. Mutation Phase

We now have a system of the form

$$\begin{cases} (M==N) \\ (M_1==N_1) \\ \hline (x \leftarrow M_2) \\ (y \leftarrow N_2) \end{cases}$$

where the non-trivial equations M==N are not decomposable. More precisely, these equations are of the form

$$t_1+t_2 == s_1*s_2$$
or
$$t_1*t_2 == s_1+s_2$$

We study in this part the transformation of these different equations into simpler Dl-equivalent systems.

First we give a result which allows us to reduce the general problem to the case where only variables occur under the top symbol left hand side of the equations, as in $S=\{x_1+x_2==t_1'*t_2'\}$. The general case is dealt with by applying an appropriate transformation called **generalisation**: For example, the equation

$t_1+t_2==t_1'*t_2'$, is solved by considering the variable case $x_1+x_2==t_1'*t_2'$ first, then by instantiating the system obtained from S by the substitution $\sigma=\{x_1\leftarrow t_1, x_2\leftarrow t_2\}$.

Lemma 5.1 — Let S_1 and S_2 be two systems of equations and σ a substitution. if $S_1 \Leftrightarrow_E S_2$ then $\sigma(S_1) \Leftrightarrow_E \sigma(S_2)$.

Proof :

Straightforward. []

Now here is the mutation process for the variable case:

Lemma 5.2 —

$$\{(x+y)==(t_1*t_2)\} \Leftrightarrow_{DI} \begin{cases}(x_2+y_2)==t_2\\ x==(t_1*x_2)\\ y==(t_1*y_2)\end{cases}$$

Proof :

Assume that σ is a solution to the equation on the left, we show that it is also a solution to the system on the right. Because of the decomposability of *, x must be of the form $x = x_1*x_2$ and $y = x_1*y_2$, thus $x+y = (x_1*x_2)+(x_1*y_2) =_{DI} x_1*(x_2+y_2)$. Hence $\sigma(x+y) =_{DI} \sigma(x_1*(x_2+y_2))$. According to the decomposability of *, we have $\sigma(x_1) =_{DI} t_1$ and $\sigma((x_2+y_2))=_{DI} t_2$.

Conversely $\sigma(x+y) = \sigma(x)+\sigma(y) = ((\sigma(x_1)*\sigma(x_2)) + (\sigma(y_1)*\sigma(y_2)) = ((t_1*\sigma(x_2)) + (t_1*\sigma(y_2)) = (t_1 * (\sigma(x_2) + \sigma(y_2)) = (t_1*t_2)$. []

We omit the proof of the next lemma, which is similar.

Lemma 5.3 —

$$\{(x*y)==(t_1+t_2)\} \Leftrightarrow_{DI} \begin{cases}(x*y_1)==t_1\\ (x*y_2)==t_2\\ y==(y_1+y_2)\end{cases}$$

5.4. The Algorithm for DI-Matching

By repeating the three steps we have a matching algorithm modulo the axiom DI. The algorithm must terminate because equations introduced during the mutation process have a size smaller than that of the equations before the mutation process.

Note, however, that this property is not true for *unification* modulo DI. In this case the mutation process constructs equations of the same type (for example $\{y*z==w+z\} \Leftrightarrow \{v_1+v_2==y*v_1,\ w==y*v_2,\ w==y*v_1\}$). In the DI matching problem, this cannot occur even if we introduce new variables during the process. The equations containing variables in their right part are equations of the trivial part and are not equations to be transformed any more.

We now give an example which illustrates the different steps of this algorithm.

Example 5.1 — Let M = (((x*y)+(z*u))+x) and N = (((a+b)*(c+d))+(a+b)). The decomposition and the merging of {M==N} gives:

$$\left\{\begin{array}{l}(x*y)+(z*u)==(a+b)*(c+d)\\ \hline x==(a+b)\end{array}\right.$$

mutation:

$$\left\{\begin{array}{l}x_2+y_2==c+d\\ x*y==(a+b)*x_2\\ z*u==(a+b)*y_2\\ \hline x==(a+b)\end{array}\right.$$

decomposition:

$$\left\{\begin{array}{l}x_2==c\\ y_2==d\\ x==(a+b)\\ y==x_2\\ z==(a+b)\\ u==y_2\\ \hline x==(a+b)\end{array}\right.$$

merging:

$$\left\{\begin{array}{l}a+b==a+b\\ \hline x_2==c\\ y_2==d\\ y==x_2\\ z==(a+b)\\ u==y_2\\ x==(a+b)\end{array}\right.$$

decomposition:

$$\left\{\begin{array}{l}\hline\\ x_2==c\\ y_2==d\\ y==x_2\\ z==(a+b)\\ u==y_2\\ x==(a+b)\end{array}\right.$$

the solution is then: {x←(a+b), y←c, z←(a+b), u←d}.

6. A Technique for Partly Eliminating Mutations.

The mutation step sometimes introduces inefficiency. We consider here a method for eliminating mutation steps by introducing a new symbol (Δ). The idea is to characterize the mutations that eventually fail because of some clash of symbols in Fex, or yield a further decomposition step.

The new symbol Δ is defined as follows:

$$x*(y+z)=\Delta(x,y,x,z)$$
$$(x*y)+(z*u)=\Delta(x,y,z,u)$$

The Knuth-Bendix completion procedure run on distributivity and these two rules returns the following canonical system ΔL:

$$x*(y+z) \rightarrow \Delta(x,y,x,z)$$
$$(x*y)+(z*u) \rightarrow \Delta(x,y,z,u)$$
$$x*\Delta(z_1,y,z,u) \rightarrow \Delta(x,z_1*y,x,z*u)$$
$$\Delta(x,y,x,z)+(z_1*u) \rightarrow \Delta(x,y+z,z_1,u)$$
$$(x*y_1)+\Delta(z_1,y,z_1,z) \rightarrow \Delta(x,y_1,z_1,y+z)$$
$$\Delta(x,y_1,x,z_2)+\Delta(z_1,y,z_1,z) \rightarrow \Delta(x,y_1+z_2,z_1,y+z)$$

Using the decomposability of + and *, it is easy to see that Δ in decomposable.

Terms are always considered in DL normal form. Every term of the forms $t_1*(t_2+t_3)$ or $(t_1*t_2)+(t_3*t_4)$ is now transformed into $\Delta(t_1',t_2',t_3',t_4')$, using ΔL, ($t_1'=t_3'=t_1$ in the first case and $t_i'=t_i$ in the second). Since Δ is decomposable, equations containing such terms are then transformed by using the decomposition process instead of a mutation process, and this is more efficient.

The mutation process is used only for those equations containing terms of the form (t_1+t_2) where t_1 and t_2 are ΔL-irreducible, or of the form (t_1*t_2) where the top symbol of t_2 is not "+".

Consequently, these transformations reduce the number of transformation steps by eliminating many mutations.

Example 6.1 — We consider the terms M and N of Example 5.1. They are transformed respectively into $\Delta(x,y,z,u)+x$ and $\Delta(a+b,c,a+b,d)+(a+b)$. Decomposition (of Δ and +) and merging steps are only used here to find solutions:
decomposition of (+) and merging:

$$\left\{\begin{array}{l}\Delta(x,y,z,u)==\Delta(a+b,c,a+b,d)\\ \hline x==a+b\end{array}\right.$$

decomposition of (Δ) and merging:

$$\left\{\begin{array}{l}\\ \hline x==a+b\\ y==c\\ z==a+b\\ u==d\end{array}\right.$$

The introduction of a new symbol Δ also allows us to reduce the equality modulo DI, to the equality of ΔL normal forms, which decreases the number of equality tests of the subterms of a term. For example for a term such as (x+y)*(z+u), the number of tests is 7 (explicitly the top symbol, +, x, y, +, z and u) whereas for its ΔL normal form $\Delta(x,y,z,u)$, the number of tests is only 5. We have thus transformed the two tests for * and + into a unique one for Δ.

Example 6.2 — Consider the following terms

$$M = (x * ((y_1 + y_2) + z))$$
$$N = (((x * y_1) + (x * y_2)) + (x * z))$$
$$M' = (x * ((y_1 + y_2) \# z))$$

The ΔL normal form of M is equal to $\Delta(x, (y_1+y_2), x, z)$ which is equal to the ΔL normal form of N. We conclude that $M =_{DI} N$. On the other hand, M' is ΔL-irreducible and the top symbols of ΔL-normal form of M and M' are not equal. This is enough to conclude that M and M' are not equal modulo DI.

7. D = DI ∪ Dr

The method used for DI to characterize the set of term modulo DI cannot be used for D = DI ∪ Dr, because all systems considered until now are not convergent (the Knuth-Bendix completion runs forever). However, matching modulo D is decidable, and we have this result:

We note $/M/_E$ the set of all terms equal to M modulo E, and $\{M==/N/_E\}$ the union $\bigcup_{N\in/N/_E} \{M==N\}_E$

Proposition 7.1 — Let A and B be two sets of axioms s.t. $=_A$ commutes with $=_B$, then we have:

$$\{M==N\}_{A\cup B} = \{M==/N/_A\}_B = \{M==/N/_B\}_A$$

Proof :
Using a commutation of $=_A$ and $=_B$ we have this equality. []

Since D is a permutative theory (admits finite equivalence classes), we can apply this result to prove decidability of D matching, and we have an algorithm to compute all matches under the axiom of distributivity.

Corollary 7.1 —

$$\{M==N\}_D = \{M==/N/_D\}_\emptyset$$

Example 7.1 —

$$\{(x+x)*y==a*(b+b)\}_D = \{(x+x)*y==/a*(b+b)/_D\}_\emptyset$$

witch is equal to:

$$\{(x+x)*y==(a*b)+(a*b)\}_\emptyset \cup \{(x+x)*y==(a+a)*b\}_\emptyset$$

The first system has an empty set of $\emptyset$-solutions (clash * and + in the empty theory), the second system is $\emptyset$-equivalent to {x←a, y←b} witch is the unique solution.

8. Conclusion

The approach used here to describe an algorithm modulo DI is based on a standard mechanism for the construction of matching algorithms, applicable to other equational theories. This mechanism is based on the following fundamental points:

- Decomposition of equations.
- Merging the constraints in order to build substitutions.
- Mutation of non-decomposable equations into decomposable equations.

In the particular case of DI, this study is simplified by introducing a new function symbol Δ with appropriate equations to define it. Δ allows us to simplify the algorithm by transforming mutation steps into decomposition steps. In the case of left distributivity, it also provides with an efficient clash for DI-equation.

9. Acknowledgement

I would like to thank the members of EURECA at CRIN. Special thanks are due to Jean-Pierre Jouannaud for many helpful suggestions. I also thank Claude Kirchner, Pierre Lescanne and Jean-Luc Remy for detailed comments and discussions during this work. The final draft of the paper is done when the author is visiting the State University of New York at Stony Brook sponsored by NCF grant DCR-8401624.

10. References

[ArT85] S. Arnborg and E. Tiden, "Unification problems with one-sided distributivity", in *Proc. 1st Conf. on Rewriting Techniques and Applications*, vol. 202, Springer Verlag, Dijon (France), 1985, 398-406.

[BeK84] J. A. Bergstra and J. W. Klop, "The Algebra of Recursively defined Processes and the Algebra of Regular Processes", in *ICALP 84*, vol. 172, 1984, 82-94.

[Der84] N. Dershowitz, "Computing With Term Rewriting Systems", *Procedings of An NSF Workshop On The Rewrite Rule Laboratory*, April 1984.

[FaH83] F. Fages and G. Huet, "Unification and Matching in Equational Theories", *Proceedings of CAAP 83*, **159**, (1983), 205-220, Springer Verlag.

[FGJ85] K. Futatsugi, J. A. Goguen, J. P. Jouannaud and J. Meseguer, "Principles of OBJ2", in *Proceedings, 12th ACM Symposium on Principles of Programming Languages Conference*, 1985.

[Hsi85] J. Hsiang, "Two Results in Term Rewriting Theorem Proving", in *Proc. 1st Conf. on Rewriting Techniques and Applications*, vol. 202, Springer Verlag, Dijon (France), 1985, 301-324.

[HuO80] G. Huet and D. Oppen, "Equations and Rewrite Rules: A Survey", in *Formal Languages: Perspectives And Open Problems*, B. R., (ed.), Academic Press, 1980.

[JoK84] J. P. Jouannaud and H. Kirchner, "Completion of a set of rules modulo a set of equations", *Proceedings 11th ACM Conference of Principles of Programming Languages*, Salt Lake City (Utah, USA), 1984.

[KaN85] D. Kapur and P. Narendran, "An Equational Approach to Theorem Proving in First-Order Predicate Calculus", in *Proc. Int. Joint Conf. on Artificial Intelligence*, Los Angeles, 1985.

[Kir85] C. Kirchner, "Méthodes et outils de conception systématique d'algorithmes d'unification dans les théories équationnelles", Thése de doctorat d'Etat, Université de Nancy I, 1985.

[Mza85] J. Mzali, "Filtrage associatif, commutatif ou idempotent", in *Proceedings of the conference Materiels et logiciels pour la 5ieme generation*, AFCET, Paris (France), 1985, 243-258.

[Sza82] P. Szabo, "Unifikationtheorie erster Ordnung", Doktorarbeit, Universitat Karlsruhe, 1982.

8.7. Exemples de filtrage par réécriture

Nous venons de voir que les opérations essentielles du filtrage (ou de l'unification [Kirchner-85]) se présentaient comme des règles de réécriture de filtrificandes. Nous exploitons ici cette idée de manière à dériver une implantation. C'est la première fois à notre connaissance qu'une telle méthode de filtrage est utilisée.

Dans cette partie, nous spécifions le mécanisme de décomposition-fusion-mutation par des équations. Nous utilisons lese symboles $\wedge$ pour la conjonction d'équations de filtrage, $\vee$ pour la disjonction, Id pour la substitution Identité et échec pour la substitution vide. Par exemple pour le cas où l'ensemble des symboles de fonctions est F = {a, b, +} avec + un symbole commutatif, nous avons le système d'équations suivant

$$(x+y)\ll(z+u) == ((x\ll z) \wedge (y \ll u)\ ) \vee ((x \ll u)\wedge(y\ll z))$$
$$(x \ll x) \ll= Id$$
$$x \wedge Id == x$$
$$a \ll b == \text{échec}$$
$$b \ll a == \text{échec}$$
$$x \wedge \text{échec} == \text{échec}$$
$$\text{échec} \vee x == x$$

La première règle est l'axiomatisation de la mutation pour ce cas particulier. les autres règles sont des règle de simplifications. La complétion de ce système complété où $\wedge$ et $\vee$ sont AC, fournit le système convergent suivant:

$(x \ll x) \rightarrow Id$
$(x \wedge Id) \rightarrow x$
$(a \ll b) \rightarrow$ échec
$(b \ll a) \rightarrow$ échec
$(x \wedge$ échec$) \rightarrow$ échec
$($échec $\vee x) \rightarrow x$
$(Id \vee Id) \rightarrow Id$
$(v_1 \wedge (x \wedge$ échec$)) \rightarrow$ échec
$(v_1 \wedge (x \wedge Id)) \rightarrow (v_1 \wedge x)$
$(v_1 \vee ($échec $\vee x)) \rightarrow (v_1 \vee x)$
$(v_1 \vee (Id \vee Id)) \rightarrow (v_1 \vee Id)$
$(Id \vee ((x \ll y) \wedge (y \ll x))) \rightarrow Id$
$(v_1 \vee (Id \vee ((x \ll y) \wedge (y \ll x)))) \rightarrow (v_1 \vee Id)$
$((x + y) \ll (z + u)) \rightarrow (((x \ll z) \wedge (y \ll u)) \vee ((x \ll u) \wedge (y \ll z)))$

Ce système convergent va nous servir à résoudre des problèmes de filtrage (et d'unification) dans $M(F,X)/=C$, en normalisant des termes dans ce système.

Par exemple pour chercher l'unificateur de $t_1 \ll (x+a)$ et $t_2 \ll (y+b)$ il suffit de normaliser le terme $(x+a) \ll (y+b)$, nous donnons une sortie de REVE 3

```
-> normal-form
Please enter the term for which you would like the normal form computed,
terminated by <ESC>:
(x+a)=(y+b)

The normal form of your term is:
((x « b) ∧ (y « a))
```

qui correspond à la substitution $\{x \leftarrow a, y \leftarrow b\}$.


```
-> nor
Please enter the term for which you would like the normal form computed,
terminated by <ESC>:
(a+b) =(y+a)

The normal form of your term is:
(b « y)

-> nor (a+b)=(b+a)

The normal form of your term is:
Id
```

9. Union de théories

9. Union de théories

Dans ce chapitre, nous étudions les transformations de théories dans lesquelles les équations sont résolues. Plutôt que de faire des transformations sur les équations, nous transformons les théories équationnelles dans lesquelles ces résolutions sont faites. Par exemple nous pouvons dire que l'équations $x+y \ll_C a+b$ dans la théorie où le symbole "+" est commutatif est équivalente au système $\{x+y \ll_\emptyset a+b,\ x+y \ll_\emptyset b+a\}$ dans la théorie vide. Nous transformons ainsi une résolution d'équations dans des théories plus simples. C'est cette approche que nous adoptons pour fabriquer des algorithmes de filtrage dans des unions de théories.

Outre les transformations de théories, nous décrivons les transformations d'équations considérées dans des unions de théories, et par conséquent nous donnons une solution au problème d'union de théories qui s'énonce comme suit : connaissant un algorithme de filtrage pour la théorie E_1, et un autre pour la théorie E_2, peut-on en déduire un algorithme de filtrage pour la théorie $E_1 \cup E_2$? Plusieurs considérations sont à prendre en compte, E_1 et E_2 sont-elles disjointes, contiennent-elles des axiomes comme l'idempotence, etc...?

C. Kirchner [Kirchner-85] et K. Yelick [Yelick-85] ont présenté une étude pour l'unification dans le cas où les deux théories ne contiennent pas les mêmes symboles, et où l'ensemble des axiomes est d'un type particulier. Nous reprenons cette étude dans le cas du filtrage, et nous montrons

certains résultats, grâce à des techniques originales dans ce cadre.

9.1. Mélange de théories disjointes

Dans ce paragraphe nous étudions un type particulier de théories, les théories qui admettent une partition séparée, c'est-à-dire les théories qui sont réunions de théories dont les ensembles de symboles de fonction sont disjoints deux à deux.

Définition 9.1 -- [Kirchner-85]-- Soient A un ensemble d'axiomes et $P=\{E_1,E_2..E_p\}$ une partition de A. Nous disons que P est une **partition séparée** de A Si et seulement si pour tout k,j de [1..p] avec $k \neq j$ on a $symb(E_k) \cap symb(E_j) = \emptyset$.

○

Notation -- Dans les conditions de la définition précédente, nous notons $A = E_1 \oplus E_2 \oplus..\oplus E_k$ lorsque P est une partition séparée de A.
Dans le cas d'une théorie équationnelle $E=E_1 \cup E_2 \cup..\cup E_n$ où chaque théorie E_i admet un système de réécriture convergent décrivant son algorithme de filtrage, si les théories E_i sont séparées, l'union des systèmes de réécriture est confluent (puisque les symboles de fonction sont disjoints) mais la terminaison du système union ne découle pas en général de la convergence des systèmes initiaux, car l'union de deux systèmes qui terminent peut ne pas terminer [Toyama-86b].

Pour avoir un système de réécriture décrivant un algorithme de filtrage pour l'union de ces théories, il suffit d'ajouter au système union

les règles d'exclusivité (les règles décrivant les symboles exclusifs). La terminaison de l'algorithme de filtrage pour l'union de théories (c'est-à-dire du système union) reste à montrer séparément.

Nous étudions maintenant une autre approche au problème du filtrage dans des unions de théories séparées basée sur des transformations de théories.

Définition 9.2 -- [Kirchner-85, Yelick-85]-- Soit P une partition séparée d'un ensemble d'axiomes A. On dit que le système d'équations S est **P-séparé** si toutes les équations eq de S sont telles qu'il existe un ensemble F_i de P tel que $symb(eq) \subseteq F_i$.

○

Lemme 9.1 -- Soit A un ensemble d'axiomes admettant une partition $P = E_1 \oplus E_2..\oplus E_p$ de théories. A toute équation eq, on peut associer un système équivalent d'équation P-séparées $E_1(eq) \wedge E_2(eq) \wedge..\wedge E_p(eq)$ telle que $sumb(E_i(eq)) \subseteq F_i$.

Preuve :

$E_i(eq)$ s'obtient par le remplacement des symboles de fonction qui ne sont pas dans E_i par des nouvelles variables en utilisant les techniques du lemme 8.6.

L'équivalence est dû au lemme 8.6

□

Lemme 9.2 -- Soit $A = E_1 \oplus E_2$ une partition séparée de théorie non potente de A.

$$M\ll_{E_1 \oplus E_2} N = E_1(M\ll_{E_1} N) \wedge E_2(M\ll_{E_2} N)$$

Preuve :

Supposons que $M=f(t_1,..,t_n)$, l'équation

$$M\ll_{E_1 \oplus E_2} N$$

est équivalente à

$$f(x_1,..,x_n)\ll_{E_1 \oplus E_2} N \wedge t_1 \leftrightarrow x_1 \wedge .. \wedge t_n \leftrightarrow x_n$$

On recommence cette opération jusqu'à ce que chaque équation eq ait ses symboles de fonction dans l'un des E_i (i=1,2) c'est-à-dire $symb(E_i(eq)) \subseteq F_i$. En regroupant les équations qui ont tous leurs symboles de fonctions dans le même ensemble F_i, nous obtenons le système suivant:

$$E_1(M\ll_{E_1 \oplus E_2} N) \wedge E_2(M\ll_{E_1 \oplus E_2} N)$$

comme $E_i(M\ll_{E_1 \oplus E_2} N)$ ne contient que des symboles de fonction de E_i, cela est équivalente à $E_i(M\ll_{E_i} N)$, d'où le résultat.

□

9.2. Mélange quelconque de théories

Etant donné un ensemble d'axiomes E, nous étudions dans ce paragraphe,

la relation entre les ensembles de solutions d'une équation $\{M\ll_E N\}$ et $\{M\ll_{E_1} N\}$ lorsque $E_1 \subseteq E$.

Notation -- Nous notons par $[N]_E$ la classe de termes égaux à N modulo E. $M\ll[N]_E$ est une notation pour la disjonction de systèmes $M\ll N_1 \vee .. \vee M\ll N_n$ pour $N_i \in [N]_E$

Définition 9.3 -- On dit que deux théories équationnelles $=_A$ et $=_B$ commutent si et seulement si $=_{AUB} = =_A =_B$

○

Le lemme suivant caractérise l'ensemble des filtres modulo AUB par rapport à celui des filtres modulo A ou modulo B.

Lemme 9.3 -- [Lemme fondamental]-- Soient A et B deux ensembles d'axiomes tels que $=_A =_B = =_B =_A$. Alors σ est un AUB-filtre de M vers N si et seulement si σ est un B-filtre de M vers $[N]_A$ (resp σ est un A-filtre de M vers $[N]_B$.

En utilisant les notations des équations, nous avons

$$M\ll_{AUB} N = M\ll_{\emptyset}[N]_{AUB} = M\ll_A[N]_B = M\ll_B[N]_A.$$

Preuve :

$$\sigma(M) =_{AUB} N \Leftrightarrow$$
$$\sigma(M) \in [N]_{AUB} \Leftrightarrow$$
$$\sigma(M) \in \{t/\ t \in [Ni]_A,\ Ni \in [N]_B\}$$

et cela à cause de la commutation de $=_A$ et $=_B$

□

9.2.1. Décidabilité de la commutation de théories

Dans le cas où deux théories sont exprimées par des systèmes de réécriture, il est possible de donner une condition suffisante pour que ces deux théories commutent.

Nous commençons par donner les définitions nécessaires.

Définition 9.4 -- Soient R et S deux systèmes de réécriture. Nous appelons c1 c2 c3 et c4 les propriétés suivantes:

$$(c1)\ \leftarrow_R \leftarrow_S \subseteq \rightarrow^*_{RUS} \leftarrow^*_{RUS} \leftarrow_R$$

$$(c2)\ \rightarrow_R \leftarrow_S \subseteq \rightarrow^*_{RUS} \leftarrow^*_{RUS} \leftarrow_R$$

$$(c3)\ \rightarrow_R \rightarrow_S \subseteq \rightarrow^*_{RUS} \leftarrow^*_{RUS} \leftarrow^R_S \rightarrow$$

où $\leftarrow^R_S \rightarrow$ est $\rightarrow_R$ ou $\leftarrow_R$ ou $\leftarrow_S$

$$(c4)\ \leftarrow_R \rightarrow_S \subseteq \rightarrow^*_{RUS} \leftarrow^*_{RUS}$$

Théorème 9.1 -- Soient R et S deux systèmes de réécriture. Supposons que

(1) R et S ont les propriétés c1 c2 c3 c4.

(2) R et S sont convergents

(3) (SUR) est convergent.

Alors $=_{RUS}$ se décompose en $=_S=_R$

Preuve :

Nous utilisons les techniques des ordres sur les preuves équationnelles, nous montrons que toutes les preuves de la formes $s=_{RUS}u$ sont supérieur à celles de la forme $s=_S=_R u$. paragraphe Rappelons (paragraphe 5.2.2) qu'un ordre sur les preuves [Bachmair,etal.-86] est basé sur un ordre de réduction sur les termes et une mesure de compléxité d'une étape de réécriture. L'ordre sur les preuves $>_P$ utilisé est basé sur la compléxité c suivante d'une 'etape (u - v):

$$c(u \rightarrow_R v) = (u, R)$$

$$c(u \rightarrow_S v) = (u, S)$$

l'ordre sur les étapes élémentaires $>_c$ est alors définie par $>_{lex}$ la combinaison lexicographique droite-gauche de

l'ordre de réécriture (RUS)

l'ordre R < S

sur les compléxites c. Par exemples $c(u \rightarrow_S v) >_c c(u \rightarrow_R$ puisque $(u,S) >_{lex} (u,R)$, puisque qu'on a R > S.

On associe à chaque preuve P le multiensemble M(?) des complexités des étapes élémentaires de preuves, d'où l'ordre $>_P$ est définie par: $P>_P P'$ ssi $M(P)>_m M(P')$, où $>_m$ est l'extention multiensemble de $>_c$.

Pour montrer que toutes les preuves par $=_{RUS}$ sont superieures au sens de l'ordre $>_P$ à des preuves du type $(\leftarrow^*\rightarrow_S)(\leftarrow^*\rightarrow_R)$, il

suffit de montrer que chaque type ($\rightarrow_R \rightarrow_S$, $\leftarrow_R \leftarrow_S$, etc..) est $>_P$ à une preuve du type $(\leftrightarrow_S)(\leftrightarrow_R)$. Les quatres types possibles sont:

(1) $\leftarrow_R \leftarrow_S$

(2) $\leftarrow_R \rightarrow_S$

(3) $\rightarrow_R \rightarrow_S$

(4) $\rightarrow_R \leftarrow_S$

Cela est vérifié grâce aux propriétés c1 c2 c3 c4 puisque

$$\leftarrow_R \leftarrow_S \; >_P \; \rightarrow^*_{RUS} \leftarrow^*_{RUS} \leftarrow_R$$

$$\rightarrow_R \leftarrow_S \; >_P \; \rightarrow^*_{RUS} \leftarrow^*_{RUS} \leftarrow_R$$

$$\rightarrow_R \rightarrow_S \; >_P \; \rightarrow^*_{RUS} \leftarrow^*_{RUS} \leftarrow^R_S \rightarrow$$

$$\leftarrow_R \rightarrow_S \; >_P \; \rightarrow^*_{RUS} \leftarrow^*_{RUS}$$

□

Nous donnons maintenant un résultat caractérisant la décidabilité des propriétés c1 c2 c3 et c4.

Théorème 9.2 -- Soient R et S deux systèmes de réécriture. Supposons que

(1) R est linéaire droit,

(2) R et S sont linéaire gauche,

(3) S satisfait la condition $V(g) = V(d)$ $\forall$ $g \rightarrow d \in S$

Alors c1, c2, c3 et c4 sont vérifiées si et seulement si les paires critiques locales vérifient c1 c2 c3 et c4

Preuve :

La preuve se fait en considérant la structure des termes. Les cas d'existence de paires critiques de réécritures sur des sous-termes disjoints, se traitent exactement comme dans [Huet-81] Nous montrons ci-dessous l'importance des conditions de linéarité pour traiter les autres cas. Nous supposons que la règle $l \rightarrow r \in R$ et $g \rightarrow d \in S$.

R est linéaire gauche est nécessaire pour éliminer les preuves du type $\leftarrow_R \leftarrow_S$ où la réécriture S a lieu sous une réécriture R (figure A).

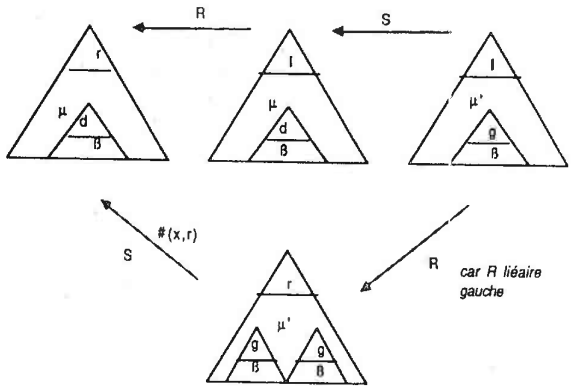


FIGURE A

S linéaire gauche est nécessaire pour éliminer les preuves du type $\rightarrow_R \rightarrow_S$ où la réécriture R a lieu sous une réécriture S (figure B).
R est linéaire droit, pour éliminer les preuves du type $\rightarrow_R \leftarrow_S$ (figure C).

□

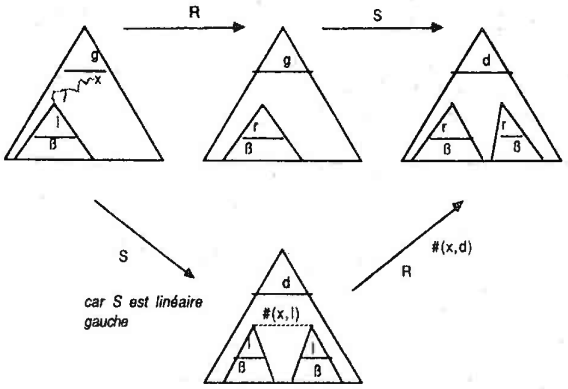


FIGURE B

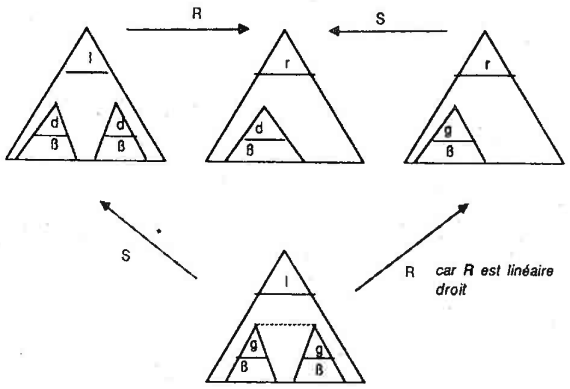


FIGURE C

Dans le cas de deux systèmes de symboles de fonctions disjoints la condition (3) du théorème 9.1 précédent devient:

(3)' (SUR) termine

car dans ce cas la réunion de deux systèmes confluents est un système confluent [Toyama-86a]. Nous avons encore besoin de la terminaison de RUS

car la réunion de deux systèmes de réécriture convergents, de symboles de fonction disjoints peut ne pas avoir la propriété de terminaison [Toyama-86b].

Exemple 9.1 -- Soit R l'ensemble d'axiomes suivants :

$$x * 1 = x$$

et soit S l'ensemble des axiomes suivants :

$$1 * x = x$$

Les systèmes de réécriture obtenus en orientant les équations de gauche à droite forment des systèmes convergents qui vérifient les hypothèses du théorème 9.1. nous avons donc

$$M\ll_{RUS}{}^{N} = M\ll_{S}[N]_{R} = M\ll_{R}[N]_{S}$$

∇

Exemple 9.2 -- Pour le cas de R = {A} et S ={C}, on ne peut pas appliquer cette méthode puisque toutes les hypothèses du théorème 9.1 ne sont pas toutes vérifiées (convergence), et dans ce cas nous perdons des solutions en séparant les deux ensembles comme en peut le voir sur l'équation $x*(y*z)\ll_{E} a*(b*c)$

∇

Le théorème 9.1 nous permet aussi de prouver la décidabilité du filtrage modulo la distributivité gauche (ou droite) et l'élément neutre: soit l'axiome de distributivité

$$Dg(*,+): x * (y + z) = (x * y) + (x * z)$$

$$Dd(*,+): (y + z) * x = (y * x) + (z * x)$$

et soit l'axiome de l'élément neutre

$$1(+): 1 + x = x$$

Pour S égale à Dg ou Dd et R égale à 1(+) ou 1(*) ou 1(+)U1(*), nous avons le résultat suivant:

$$M\ll_{R\cup S}N = M\ll_S[N]_R = M\ll_R[N]_S$$

car dans ce cas, les systèmes R et S correspondants sont convergents et vérifient les hypothèses du théorème 9.1.

9.2.2. Ramener le filtrage modulo E au filtrage dans la théorie vide

Les résultats du paragraphe précédent nous permettent aussi de ramener le problème de filtrage dans une théorie E en une réunion de problèmes de filtrage dans la théorie vide, nous avons :

$$M\ll_E N = M\ll_\emptyset[N]_E$$

En effet, les axiomes de la théorie vide (aucun axiome) commutent avec tous les autres axiomes.

Ce résultat nous donne une méthode générale de mutation de filtricandes. Il nous permet en particulier de prouver la décidabilité du filtrage modulo la distributivité (gauche et droite) [Mzali86]. A titre de comparaison, la décidabilité de l'unification distributive est un problème ouvert.

Exemple 9.3 -- Soit D l'ensemble qui contient les axiomes de distributivité (gauche Dg(*,+) et droite Dd(*,+))

Nous avons

$$(x+x)*y\ll_D a*(b+b) =$$
$$(x+x)*y\ll_\emptyset [a*(b+b)]_D =$$
$$(x+x)*y\ll_\emptyset a*(b+b) \vee$$
$$(x+x)*y\ll_\emptyset (a*b)+(a*b) \vee$$
$$(x+x)*y\ll_\emptyset (a+a)*b$$

D'où la solution donnée par le troisième système $\{x\leftarrow a,\ y\leftarrow b\}$, tous les autres systèmes n'ayant pas de solutions dans la théorie vide.

∇

Nous pouvons aussi utiliser cette méthode pour transformer le filtrage associatif-commutatif $M\ll_{AC}N$, en un filtrage dans la théorie vide $M\ll_\emptyset[N@sub"[",AC]$. Cette méthode nous semble interessante dans le cas de recherche d'une seule solution de filtrage.

Nous donnons maintenant un résultat qui permet de construire un algorithme de filtrage pour un type particulier de théories, les théories finies (chaque classe d'équivalence est finie)

9.2.3. Filtrage modulo E union une théorie finie

Corollaire 9.1 -- Soit A une théorie finie et soit B une théorie pour laquelle un algorithme de filtrage est connu. Si les ensembles d'axiomes A et B vérifient les hypothèses du théorème 9.1, alors le filtrage dans la

théorie AUB est décidable. Un algorithme pour le filtrage est donné par :

$$M \ll_{AUB}^{N} = M \ll_{B} [N]_{A}$$

Ce qui veut dire qu'on cherche d'abord les éléments Ni égaux modulo A à N, (qui sont en nombre fini puisque la théorie A est une théorie finie), et on applique l'algorithme de filtrage de la théorie B sur les équations $M \ll_{E} Ni$ ensuite.

9.2.4. Caractérisation de la Mutation

Une autre application de ce qui précède consiste à décrire la phase de mutation d'un algorithme de filtrage modulo E. Par exemple pour le cas de commutativité, la mutation peut être décrite en ramenant la résolution d'un filtricande modulo C à la résolution dans la théorie vide, ce qu'on peut exprimer ainsi pour le cas de ces deux termes:

$$f(x,y) \ll_{C(f)} f(z,u) \Leftrightarrow f(x,y) \ll_{C(f)} [f(z,u)]_{C(f)}$$
$$\Leftrightarrow (x \ll_{\emptyset} z \wedge y \ll_{\emptyset} u) \vee (x \ll_{\emptyset} u \wedge y \ll_{\emptyset} z)$$

ce qui correspond à la règle de réécriture suivante:

$$f(x,y) \ll_{C(f)} f(v,u) \rightarrow (x \ll_{\emptyset} z \wedge y \ll_{\emptyset} u) \vee (x \ll_{\emptyset} u \wedge y \ll_{\emptyset} z)$$

Dans le cas général $(f(t_1,t_2) \ll_{C(f)} f(t_1',t_2'))$ et $f \in symb(t_1) \cup symb(t_2))$ la règle de mutation est la suivante:

$$f(x,y) \ll_{C(f)} f(v,u) \rightarrow$$
$$(x \ll_{C(f)} z \wedge y \ll_{C(f)} u) \vee (x \ll_{C(f)} u \wedge y \ll_{C(f)} z)$$

Notons que le processus de mutation termine dans ce cas parce que le système de réécriture composé de la seule règle précédente (avec $\wedge$ et $\vee$ symbole AC) est convergent.

CONCLUSION

Il est possible de dégager de cette thèse deux types d'apport:

► Dans le domaine de la démonstration automatique.

• la conception d'un algorithme de complétion qui prévilégie la règle de simplification.

• l'étude et l'implantation de deux types de preuves basées sur des extensions de l'algorithme de complétion, à savoir les stratégies N et RN et UKB.

Cette étude permet de voir que les méthodes de démonstration automatique basées sur des extensions de l'algorithme de complétion, manipulent toujours les même outils de base comme par exemple l'unification, la superposition, la simplification. On voit aussi que pour passer d'une méthode à une autre, il suffit parfois de remplacer un algorithme de E-unification par un autre de E'-unification (complétion modulu E), ou de modifier la méthode de calcul des paires critiques (N- et RN-strategy). Pour avoir un

système de démonstration puissant et souple, il est nécessaire de disposer d'un certains nombres d'outils de base, qui permettent d'implanter une méthode ou une autre suivant la manière de combiner ces outils. Des systèmes tels que REVE 3 ou ERIL [Dick-85] offrent certaines possibilités: REVE 3 permet de programmer sa propre théorie équationnelle en ajoutant au système l'algorithme d'unification et de filtrage de cette théorie, ERIL permet à l'utilisateur de définir ses ensembles de règles et de paires de termes à orienter en spécifiant quelles superpositions et réductions sont nécesssaires pour chacun. En regroupant toutes ces idées, on peut voir la nécessité d'un langage de haut niveau, permettant de maniputer facilement, les différentes unifications, les strategies de superpositions, les systèmes de règles et d'équations etc..

▶ Dans le domaine du filtrage équationnel.

• Nous avons spécifié les principales opérations d'un algorithme de filtrage équationnel à l'aide de règles simples qui sont en général des règles de réécriture.

• Nous avons adopté une approche pour construire des algorithmes de filtrage différente de celle définie pour l'unification par C. Kirchen, K. Yelick, E. Tiden ou A. Herold, à savoir la transformation de théories. Cette méthode nous a permi de donner un algorithme de filtrage pour l'axiome de distributivité gauche et droite, et pour l'axiome de commutativité droite.

Les transformation de théories peuvent nous ammener à des fusions de filtricandes du type $(x \ll_{E_1} t_1 \wedge x \ll_{E_2} t_2)$, ce type de fusion qui généralise la notion classique de fusion n'est pas étudié dans cette thèse, il est encore

un problème ouvert.

D'autre part, la combinaison d'algorithmes décrits par des systèmes de réécriture est une possibilité attrayante, quoique la terminaison de la réunion des deux systèmes soit à prouver dans chaque cas.

Enfin, l'indépendance de la résolution des filtricandes dans notre méthode, ouvre des voies dans la définition d'algorithmes parallèles pour le filtrage équationnel.

ANNEXE

Théories utilisées

Nous adoptons la notation suivant: si R est un ensemble d'équations $R\downarrow$ est le système convergent des règles de réécritures obtenu par l'algorithme de complétion de Knuth-Bendix ou UKB.

group

```
(e * x) == x
(i(x) * x) == e
((x * y) * z) == (x * (y * z))
```

group↓

$$
\begin{aligned}
& e * x \rightarrow x \\
& i(x) * x \rightarrow e \\
& (x * y) * z \rightarrow x * (y * z) \\
& i(y) * (y * z) \rightarrow z \\
& i(e) \rightarrow e \\
& z * e \rightarrow z \\
& i(i(z_1)) \rightarrow z_1 \\
& z_1 * i(z_1) \rightarrow e \\
& x * (i(x) * z) \rightarrow z \\
& i(y_1 * y) \rightarrow i(y) * i(y_1)
\end{aligned}
$$

fib

```
(0 + x) == x
(s(x) + y) == s((x + y))
((x + y) + z) == (x + (y + z))
fib(0) == 0
fib(s(0)) == s(0)
fib(s(s(x))) == (fib(x) + fib(s(x)))
dfib(0, y) == y
dfib(s(0), y) == s(y)
dfib(s(s(x)), y) == dfib(s(x), dfib(x, y))
```

fib↓

```
fib(0) → 0
0 + x → x
dfib(0, y) → y
fib(s(0)) → s(0)
dfib(s(0), y) → s(y)
s(x) + y → s(x + y)
(x + y) + z → x + (y + z)
fib(s(s(x))) → fib(x) + fib(s(x))
dfib(s(s(x)), y) → dfib(s(x), dfib(x, y))
```

ackermann

```
a(0,x) == s(x)
a(s(x),0) == a(x,s(0))
a(s(x),s(y)) == a(x,a(s(x),y))
```

ackermann↓

```
a(0, x) → s(x)
a(s(x), 0) → a(x, s(0))
a(s(x), s(y)) → a(x, a(s(x), y))
```

stack

```
if_then_else(true, x, y) == x
if_then_else(false, x, y) == y
is_empty(empty_stack) == true
is_empty(push(u, x)) == false
top(empty_stack) == error_element
top(push(u, x)) == x
pop(push(u, x)) == u
pop(empty_stack) == error_stack
```

stack↓


```
is_empty(empty_stack) → true
top(empty_stack) → error_element
pop(empty_stack) → error_stack
if_then_else(true, x, y) → x
if_then_else(false, x, y) → y
is_empty(push(u, x)) → false
top(push(u, x)) → x
pop(push(u, x)) → u
```

z

```
_(0) == 0
_(_(x)) == x
(x + 0) == x
(0 + x) == x
((x + y) = (x + z)) == (y = z)
((x + y) = (z + x)) == (y = z)
((y + x) = (x + z)) == (y = z)
((y + x) = (z + x)) == (y = z)
((x + y) = x) == (y = 0)
((y + x) = x) == (y = 0)
(x = (x + y)) == (y = 0)
(x = (y + x)) == (y = 0)
(x = 0) == nulle(x)
(x = x) == true
(_(x) + _(y)) == _((x + y))
(_(x) = _(y)) == (x = y)
(0 * x) == 0
(1 * x) == x
(_(x) * y) == _((x * y))
(x - y) == (x + _(y))
((x + y) * z) == ((x * z) + (y * z))
(0 + x) == x
```

z↓

```
_(0) → 0
nulle(0) → true
_(_(x)) → x
x + 0 → x
0 + x → x
1 * x → x
0 * x → 0
x = x → true
nulle(_(y)) → nulle(y)
0 = y → nulle(y)
x = 0 → nulle(x)
x = (x + y) → nulle(y)
x = (y + x) → nulle(y)
(x + y) = x → nulle(y)
(y + x) = x → nulle(y)
x = _(y) → _(x) = y
_(x) * y → _(x * y)
_(_(y) + y1) = y → nulle(y1)
x + _(y) → _(_(x) + y)
(y + x) = (z + x) → y = z
(y + x) = (x + z) → y = z
(x + y) = (z + x) → y = z
(x + y) = (x + z) → y = z
(x + y) * z → (x * z) + (y * z)
_(y + x) = (_(x) + y1) → _(y) = y1
_(_(x) + y) = (z + x) → _(y) = z
_(_(x) + y) = (x + z) → _(y) = z
_(x + y) = (_(x) + y1) → _(y) = y1
```

pgl

```
f(x,x) == 0
f(x,f(y,z)) + f(y,f(z,x)) + f(z,f(x,y)) == 0
f(x,f(y,z)) == 0
f(f(x,f(y,z)),u) == 0
f(f(f(x,f(y,z)), u), v) == 0
f(f(f(f(x,f(y,z)), u), v),w) == 0
```

pgl↓

```
f(x, x) → 0
f(0, u) → 0
f(x, 0) → 0
f(x, f(y, z)) → 0
0 + (0 + 0) → 0
```


■

```
(x.y).(z.w) = (x.z).(y.w)
    (x.y).x = x
```

■↓

```
(x.y).x → x
x.(y.z) → x.z
(x.y).z = (x.w).z
((x.y).z).w →x.w
```

poids@

```
(x.y).x = x
x.(y.z) = x.z
(x.y).z = f(x,z)
f(x,z).w = x.w
((x.y).z).w = x.w
```

poids↓

```
f(x, x) → x
x . (y . z) → x . z
(x . y) . z → f(x, z)
f(x, z) . w → x . w
f(x . y, w) = x . w
f(x, y . z) → f(x, z)
y . f(x, z) → y . z
f(f(x, z), y) → f(x, y)
f(x, f(y, z)) → f(x, z)
f(x . z, w) = f(x . y, w)
```

References

Arnborg-Tiden-85.

S. Arnborg and E. Tiden, "Unification problems with one-sided distributivity," in Proc. 1st Conf. on Rewriting Techniques and Applications, Lecture Notes in Computer Science, vol. 202, pp. 398-406, Springer Verlag, Dijon (France), 1985.

Bachmair,etal.-86.

L. Bachmair, N. Dershowitz, and J. Hsiang, "Orderings for Equational Proofs," in Proc. Symp. Logic in Computer Science, Cambridge, Massachussetts, June 1986.

Bachmair-Dershowitz-86a.

L. Bachmair and N. Dershowitz, "Commutation, Transformation, and Termination," in Proc. 8th Conf. on Automated Deduction, Lecture Notes in Computer Science, vol. 230, Springer Verlag, Oxford (England), 1986.

Bachmair-Dershowitz-86b.

L. Bachmair and N. Dershowitz, Critical pair criteria for the Knuth-Bendix completion method, University of Illinois, Urbana-Champaign, Illinois, 1986.

Baxter-73.

L. D. Baxter, "An Efficient Unification Algorithm," Technical Report Cs-73-23, Dept. of Applied Analysis And Computer Science U. of Waterloo, 1973.

BenCherifa-86.

A. BenCherifa, "Preuves de terminaison de systèmes de réécriture fondées sur des interprétations polynomiales," Thèse, Université de Nancy 1, Nancy (France), 1986.

BenCherifa-Lescanne-86.

A. BenCherifa and P. Lescanne, "An actual implementation of a procedure that mechanically proves termination of rewriting systems based on inequalities between polynomial interpretations," in Proc. 8th Conf. on Automated Deduction, Lecture Notes in Computer Science, vol. 230, Springer Verlag, Oxford (England), 1986.

Bourbaki-68.

N. Bourbaki, Eléments de mathématique. Théorie des ensembles, Hermann, Paris, 1968.

Burckert-86.

H. Burckert, "Some Relationships between Unification, Restricted Unification, and Matching," in Proc. 8th Conf. on Automated Deduction, Lecture Notes in Computer Science, vol. 230, Springer Verlag, Oxford (England), 1986.

Burckert-Herold-86.

H. Burckert, A. Herold, and M. Schmidt-Schaub, "On Equational Theorie, Unification And Decidability," Universitat Kaiserslautern, 1986.

Corbin-Bidoit-83.

J. Corbin and M. Bidoit, "Pour une réhabilitation de l'algorithme d'unification de Robinson," IFIP 1983, 1983.

Dershowitz-82.

Nachum Dershowitz, "Orderings for Term-Rewriting Systems," Theoretical Computer Science, vol. 17, pp. 279-301, 1982.

Dershowitz-84.

N. Dershowitz, "Computing With Term Rewriting Systems," Procedings of An NSF Workshop On The Rewrite Rule Laboratory, April 1984.

Dershowitz-85.

N. Dershowitz, "Termination," in Proc. 1rst Conf. Rewriting Techniques and Applications, Lecture Notes in Computer Science, vol. 202, pp. 180-224, Springer Verlag, Dijon (France), May 1985.

Dershowitz-Manna-78.

N. Dershowitz and Z. Manna, "Proving Termination With Multiset Orderings," Comm. of ACM, vol. 22, pp. 465-476, 1979.

Dick-85.

A.J.J. Dick, "ERIL - Equational reasoning: an interactive laboratory," Proceedings of the EUROCAL Conference, Linz (Austria), 1985.

Durand-82.

J. Durand, "LOGRE: Un prototipe de système de programmation en logique utilisant des techniques de réécriture," Thèse de troisième cycle, Univ. de Nancy-I, CRIN T-82, 1984.

Fages-83.

F. Fages, "Formes canoniques dans les algèbres booléennes," Thèse de 3eme cycle. Université de Paris 6, 1983.

Fages-Huet-83.

F. Fages and G. Huet, "Unification and Matching in Equational Theories," Proceedings of CAAP 83, vol. 159, pp. 205-220, Springer Verlag, l'Aquilla, Italy, 1983.

Forgaard-84.

R. Forgaard, "A program for generating and analysing term rewriting systems," MIT/LCS/TR-343, 1984.

Forgaard-Guttag-84.

R. Forgaard and J.V. Guttag, "REVE: A Term Rewriting System Generator with Failure-Resistant Knuth-Bendix," MIT-LCS, 1984.

Fribourg-83.

L. Fribourg, "A Superposition Oriented Theorem Prover," Proceedings 10th IJCAI, 1983.

Futatsugi,etal.-85.

K. Futatsugi, J.A. Goguen, J.P. Jouannaud, and J. Meseguer, "Principles of OBJ2," in Proceedings, 12th ACM Symposium on Principles of Programming Languages Conference, 1985.

Gnaedig-86.

I. Gnaedig, "Preuves de Terminaison des Systemes de Réécriture Associatifs-commutatifs: une Methode Fondee sur la Reecriture Ellememe," These de troisieme cycle, Nancy, 1986.

Goguen,etal.-82.

J.A. Goguen, J. Meseguer, and D. Plaisted, "Programming with Parameterized Abstract Objects in OBJ.," in Theorie And Practice of

Software Technology, pp. 163-193, North-Holland, 1982.

Goguen-80.

J.A. Goguen, "How to Prove Algebraic Inductive Hypotheses Without Induction, With Applications to the Correctness of Data Type Implementation," in Proc. 5th Workshop on Automated Deduction, ed. W. Bibel and Kowalski, Lecture Notes in Computer Science, vol. 87, pp. 356-373, Springer Verlag, Les Arcs, France, 1980.

Henkin-84.

L. Henkin, "The Logic of Equality," Mathematical Monthly, vol. 84, pp. 597-612, October 1977.

Herold-86.

A. Herold, "Combination of unification algorithms," in Proc. 8th Conf. on Automated Deduction, Lecture Notes in Computer Science, vol. 230, pp. 450-469, Springer Verlag, Oxford (England), 1986.

Hsiang-82.

J. Hsiang, "Topics in Automated Theorem Proving And Program Generation," PHD Thesis Unv. of Illinois at Urbana-Campaign, 1982.

Hsiang-Dershowitz-83.

J. Hsiang and N. Dershowitz, "Rewrite methods for clausal and non clausal theorem proving," Proceedings 10th Colloquium in Automata Languages and Programming, vol. 154, Barcelona, Spain, 1983.

Hsiang-Josephson-83.

J. Hsiang and N.A. Josephson, "TeRSe: a Term Rewriting Theorem Prover," Univ. of Illinois at Urbana-Champaign, 1983.

Hsiang-Plaisted-82.

J. Hsiang and D. Plaisted, "Deductive Program Generation," Technical Report, University of Illinois, Computer Science Departement, 1982.

Hsiang-Rusinowitch-85a.

J. Hsiang and M. Rusinowitch, "A New Method For Establishing Refutational Completeness in Theorem Proving," Ten. Rep. 85/24, SUNY at Stony Brook, 1985.

Hsiang-Rusinowitch-85b.

J. Hsiang and M. Rusinowitch, "On Word Problems in Equational Theories," CRIN 85-R-112, 1985.

Huet-76.

G. Huet, "Résolution d'Equations dans des Langages d'Ordre 1,2, ... ω," Thèse d'Etat, Université de Paris VII, 1976.

Huet-81.

G. Huet, "A complete proof of correctness of the Knuth-Bendix completion algorithm," J. Comp. Sys. Sc., vol. 23, no. 1, pp. 11-21, Aug. 1981.

Huet-Hullot-80.

G. Huet and J.M. Hullot, "Canonical Form Algorithms for Finitely Presented Algebras," Working Paper, 1980.

Huet-Hullot-82.

G. Huet and J.M. Hullot, "Proofs By Induction in Equational Theories With Constructors," J. Comp. Sys. Sc., vol. 25, pp. 239-266, 1982.

Huet-Lankford-78.

G. Huet and D.S. Lankford, "On the Uniform Halting Problem for Term Rewriting Systems," Rapport Laboria 283, Iria, Mars 1978.

Hullot-80.

J.M. Hullot, "Compilation de Formes Canoniques dans les Théories Equationnelles," Thèse de 3ème Cycle, Université de Paris Sud, 1980.

Hullot-80a.

J.M. Hullot, "Canonical Forms And Unification," in Proceedings of the Fifth Conference on Automated Deduction, Lecture Notes in Computer Science, vol. 87, pp. 318-334, Springer Verlag, Les Arcs, France, July 1980.

Jouaaud-83.

J.P. Jouannaud, "Church-Rosser computations with equational term rewriting systems," to appear in J. ACM, 1983.

Jouannaud,etal.-82.

J.P. Jouannaud, P. Lescanne, and F. Reinig, "Recursive Decomposition Ordering," in Formal Description of Programming Concepts 2, ed. Bjorner D., pp. 331-346, North Holland, Garmish Partenkirschen, RFA, 1982.

Jouannaud,etal.-83.

J. P. Jouannaud, C. Kirchner, and H. Kirchner, "Incremental Construction of Unification Algorithms in Equational Theories," in Proceedings of the International Conference On Automata, Languages and Programming, Lecture Notes in Computer Science, vol. 154, pp. 361-373,

Springer Verlag, Barcelona Spain, 1983.

Jouannaud-83.

J.P. Jouannaud, "Confluent and coherent equational term rewriting systems. application to proofs in abstract data types," Proceeding of the 8th colloquium on trees an algebra and programming, L'Aquila, 1983.

Jouannaud-Kirchner-84.

J.P. Jouannaud and H. Kirchner, "Completion of a set of rules modulo a set of equations," Proceedings 11th ACM Conference of Principles of Programming Languages, Salt Lake City, 1984.

Jouannaud-Kirchner-86.

J.Jouannaud and H.Kirchner, "Completion of a set of rules modulo a set of equations," SIAM J. of Computing, vol. 15(4), 1986.

Jouannaud-Kounalis-86.

J.P. Jouannaud and E. Kounalis, "Proofs by induction in equational theories without constructors," Proc. 1rst IEEE Symp. on Logic in Computer Science, Cambridge, Massachusetts, (USA), 1986.

Jouannaud-Lescanne-82.

J. P. Jouannaud and P. Lescanne, "On Multiset Orderings," Information Processing Letters, vol. 10, no. 2, pp. 57-63, 1982.

Joyal-81.

A. Joyal, Une Théorie Combinatoire des Séries Formelles, Academic Press Inc, 1981.

Kapur,etal.-85.

D. Kapur, D. Musser, and P. Narendran, Only prime superpositions need be considered in the Knuth-Bendix procedure, General Electric Corporate Research and Development, Schenectady, New-York, 1985.

Kirchner-85.

C. Kirchner, "Méthodes et outils de conception systématique d'algorithmes d'unification dans les théories équationnelles," Thèse de doctorat d'Etat, Université de Nancy I, 1985.

Kirchner-85b.

H. Kirchner, "Preuves par complétion dans les variétés d'algèbres," Thèse de doctorat d'Etat, Université de Nancy I, 1985.

Kirchner-Kirchner-85.

C. Kirchner and H. Kirchner, "Implementation of a general completion procedure parameterized by built-in theories and strategies," Proceedings of the EUROCAL `85 conference, vol. 2, pp. 402-404, 1985.

Knuth-Bendix-70.

D. Knuth and P. Bendix, "Simple Word Problems in Universal Algebras," Computational Problems in Abstract Algebra Ed. Leech J., Pergamon Press, pp. 263-297, 1970.

Kuechlin-85.

W. Kuechlin, "A confluence criterion based on the generalized Newman lemma," Proceedings of the EUROCAL Conference, Linz (Austria), 1985.

Liskov,etal.-81.

Barbara Liskov, Eliot Moss, Craig Schaffert, Bob Scheifler, and Alan Snyder, Clu Reference Manual, Lecture Notes in Computer Science, 114,

1981.

Loveland-78.

D.W. Loveland, Automated Theorem Proving: a Logical Basis, 1978.

Martelli-Montanari-82.

A. Martelli and U. Montanari, "An efficient unification algorithm," ACM T.O.P.L.A.S, vol. 4, no. 2, pp. 258-282, 1982.

Musser-80.

D. L. Musser, "On Proving Inductive Properties of Abstract Data Types," Proceedings of the 7th Annual Acm Symposium on Principles of Programming Languages, pp. 154-162, Las Vegas, 1980.

Mzali-84.

J. Mzali, "Filtrage Associatif, Commutatif ou Idempotent et ses Preuves," Rapport Crin 85-R-41, 1984.

Mzali-85.

J. Mzali, "Filtrage associatif, commutatif ou idempotent," in Proceedings of the conference "Hardware and Sofware Components and Architectures for the 5th Generation", pp. 243-258, Paris (France), 1985.

Mzali-86b.

J. Mzali, "Matching With Distributivity," in Proc. 8th Conf. on Automated Deduction, Lecture Notes in Computer Science, vol. 230, pp. 497-505, Springer Verlag, Oxford (England), 1986.

Mzali1983.

Jalel Mzali, "Algorithme de Filtrage Associatif Commutatif," Rapport

de Dea. Crin 83-R-60, juin 1983.

Paul-84.

E. Paul, "A new interpretation of the resolution principle," Proceedings 7th international Conference on Automated Deduction, vol. 170, Napa Valley (California, USA), 1984.

Paul-84.

E. Paul, "Proof by induction in equational theories with relations between constructors," in Proc. 9th Colloquium on trees in Algebra and Programming, ed. B. Courcelle, pp. 210-225, Cambridge University Press, Bordeaux, 1984.

Peterson-83.

G.E. Peterson, "A Technique for Establishing Completeness Results in Theorem proving with equality," J.ACM, vol. 28, pp. 233-264, 1983.

Peterson-Stickel-81.

G. Peterson and M. Stickel, "Complete sets of reduction for equational theories with complete unification algorithms," J. of ACM, vol. 28, no. 2, pp. 233-264, 1981.

Plotkin-72.

G. Plotkin, "Building-In Equational Theories," Machine Intelligence, vol. 7, pp. 73-90, 1972.

Raoult-Vuillemin-78.

J.C. Raoult and J. Vuillemin, "Operational And Semantic Equivalence Between Recursive Programs," 10Th Acm Symposium on Theory of Computing, San Dieg, 1978.

Remy-82.

J.L. Remy, "Etude des systèmes de Réécriture Conditionnelle et Applications aux Types Abstraits Algébriques," Thèse d'Etat, I.N.P.L., Nancy, 1982.

Remy-Zhang-84.

J.L. Remy and H. Zhang, "REVEUR 4: a System for Validating Conditional Algebraic Specifications of Abstract Data Types," Proceedings of the 5th ECAI, Pisa, 1984.

Rety,etal.-85.

P. Rety, C. Kirchner, H. Kirchner, and P. Lescanne, "NARROWER: A new Algorithm for Unification and its application to Logic Programming," in Proc. 1rst Conf. on Rewriting Techniques and Applications, Lecture Notes in Computer Science, vol. 202, pp. 141-157, Springer Verlag, Dijon (France), 1985.

Rety-87.

P. Réty, "Improving basic narrowing techniques," 2nd Conf. on Rewriting Techniques and Applications, Bordeaux, 1987.

Robinson-65.

J.A. Robinson, "A Machine-Oriented Logic Based on the Resolution Principle," J. of ACM, vol. 12, pp. 32-41, 1965.

Robinson-71.

J.A. Robinson, "Computational Logic: the Unification Computation," Machine Intelligence 6, Eds B. Meltzer And D.Michie American Elsevier, New-York, 1971.

Sethi-74.

R. Sethi, "Testing for the Church-Rosser Property," J. of ACM, vol. Jacm 22 P. 424, pp. 671-679, 1974.

Shoenfield-67.

J.R. Shoenfield, Mathematical Logic, 1967.

Siekmann-78.

J. Siekmann, "Unification And Matching Problems," Ph. D. Thesis, Memo Csm-4-78, U. of Essex, 1978.

Stone-36.

M. Stone, "The Theory of Representation for Boolean Algebras," Trans. Ams Vol. 40, pp. 37-111, 1936.

Szabo-82.

P. Szabo, "Unifikationtheorie erster Ordnung," Doktorarbeit, Universitat Karlsruhe, 1982.

Tiden-86.

E. Tiden, "Unification in combinations of collapse-free theories with disjoint sets of fonction symbols," in Proc. 3th Conf. on Automated Deduction, Lecture Notes in Computer Science, vol. 230, pp. 431-449, Springer Verlag, Oxford (England), 1986.

Toyama-86a.

Y. Toyama, "On the Church-Rosser property for the direct sum of term rewriting systems," Journal of the ACM, to appear, 1986.

Toyama-86b.

Y. Toyama, "Contrexamples to termination for the direct sum of term rewriting systems," Private comunication, 1986.

Winkler-Buchberger-83.

F. Winkler and B. Buchberger, "A criterion for eliminating unnecessary reductions in the Knuth-Bendix algorithm," Colloquium on Algebra, Combinatorics and Logic in Computer Science, Gyor (Hungary), 1983.

Yelick-85.

K. Yelick, "Combining unification algorithms for confined equational theories," in Proc. 1st Conference on Rewriting Techniques and Applications, Lecture Notes in Computer Science, vol. 202, pp. 365-380, Springer Verlag, Dijon (France), 1985.

NOM DE L'ETUDIANT : MZALI Jalel

NATURE DE LA THESE : Doctorat de l'Université de NANCY I en Informatique

VU, APPROUVE ET PERMIS D'IMPRIMER 2384

NANCY, le 15 OCT. 1986

LE PRESIDENT DE L'UNIVERSITE DE NANCY I

R. MAINARD

UNIVERSITÉ DE NANCY I Le Président

RESUME

L'objet de la première partie de cette thèse est l'étude et l'implantation de différentes méthodes de démonstration automatique basées essentiellement sur un algorithme de complétion rapide appelé SKB et un algorithme de complétion sans échec appelé UKB. SKB est un algorithme de complétion qui privilégie la règle de simplification par rapport à celle de superposition, nous étudions cet algorithme et son implantation, nous prouvons sa correction totale par la technique des *ordres sur les preuves équationnelles* de Bachmair-Dershowitz-Hsiang. L'algorithme UKB de complétion sans échec, dû à Hsiang-Rusinowitch est étudié et implanté, il nous permet de construire des preuves dans les théories équationnelles et les théories équationnelles généralisées.

La deuxième partie de cette thèse est consacrée à l'étude du problème de filtrage qui est crucial pour la simplification et la réécriture des termes. Ce problème est étudié d'un point de vue algébrique, ce qui nous permet d'exprimer les principales phases des algorithmes de filtrage par des règles d'inférence simples dont la traduction sous forme d'un système de réécriture fournit un algorithme de filtrage. Cette étude algébrique nous permet de donner un algorithme original dans le cas du filtrage modulo la distributivité et par conséquent de prouver la décidabilité du filtrage distributif. Un algorithme pour le filtrage modulo l'axiome de commutativité droite est également donné. De la même manière que les transformations d'équations, nous définissons les transformations de théories dans lesquelles les équations sont résolues. Dans le cas d'union de théories E$\cup$T, nous donnons des conditions suffisantes pour ramener le problème de E$\cup$T-filtrage d'un système d'équations au problème de E-filtrage (ou T-filtrage) d'un autre système d'équations.

Mots clefs: Théorie équationnelle, preuve automatique, preuve par réfutation, algorithme de complétion, algorithme de complétion sans échec, filtrage équationnel, décomposition, fusion, mutation.